ZOE: UN SISTEMA DE MENSAJERÍA F2F

Carlos Enrique Novo Negrillo

Sevilla, Septiembre 2013

Contents

Introducción

Objetivos

Objetivos Generales

Se pretende desarrollar el software para un nodo que sea capaz de comunicarse punto a punto con otros nodos de confianza.

Si bien el presente trabajo puede servir como base para desarrollar una red anónima y descentralizada tipo Freenet o Turtle, queda fuera del alcance del mismo. Queda a disposición de futuros trabajos el extender el mismo para dotarle de mayores funcionalidades.

El que, como se mostrará posteriormente, existan múltiples desarrollos que contemplan muchas de las características de nuestra propuesta, esto no descalifica el presente trabajo por considerarse que se trata de un campo en continuo desarrollo cuya aplicación será habitual en las comunicaciones de los próximos años.

Incluso Retroshare, que incluye dos de las principales características deseadas: F2F a través de NATs y descentralización, convierte una de sus características: ausencia total de servidores, en una desventaja, ya que al depender de redes DHT la localización de los nodos tarda mucho tiempo o llega incluso a no tener éxito.

En este sentido, las características deseables de dicho nodo son:

- Debe ser *GUI independent*
- No debe requerir que el usuario final configure ningún tipo de firewall o redireccionamiento.
- Debe tener canales de actuación, al menos una consola telnet y un canal TCP para manejarlo.
- Debe ser multiplataforma
- En este trabajo, el alcance será *Poder enviar un mensaje encriptado a un contacto sin que pase por ningún tercero*.
- Esta funcionalidad básica deberá ser fácilmente extensible para poder transmitir cualquier tipo de contenido.
- Si bien, otros desarrolladores podrán tener acceso a los fuentes y extenderlos directamente, lo expuesto en el canal TCP deberá ser suficiente para poder extender la funcionalidad aunque no se tuviera acceso a fuentes ni API.

Objetivos Específicos

- En cuanto al software:

  Se debe diseñar y construir un software que implemente un nodo que:

  - Una vez instalado, el nodo debe ser capaz de arrancar y funcionar tan sólo definiendo en la configuración un id único de usuario (ej. email) y la dirección de, al menos, un presentador.
  - Debe ser capaz de localizar otros nodos en internet o en LAN
  - Debe ser capaz de enviar mensajes a otros nodos, sin importar dónde están, qué IP tienen o en qué puerto están accesibles.
  - Debe ser capaz de recibir mensajes de otros nodos sin que estos sepan dónde están ni tras qué topología de red.
  - Toda la información que necesita un nodo para comunicarse con otro será un UUID del otro nodo, por ejemplo, una dirección de correo electrónico.
  - Todas las actividades que ejecute el nodo deberán ser, siempre que sea posible, asíncronas.
  - Debe tener una consola de debug por telnet, que dé acceso al código ejecutándose, con objeto de debug.
  - Debe tener un canal TCP para que aplicaciones de terceros se conecten a él y lo manejen a modo de *core*
  - Debe ser *plugeable*. Se podrán añadir fácilmente plugins escritos por terceros.
  - Debe soportar encriptación fuerte por defecto.
  - Debe poder manejarse desde una consola telnet, con un repertorio suficiente de comandos con los que:

    - Invitar a contacto
    - Aceptar invitación
    - Eliminar contacto
    - Enviar un mensaje, plano o encriptado, a contacto
    - Acceder a los mensajes recibidos
    - Acceder al historial y estado de mensajes enviados

- En cuanto a metodología:

  - Profundizar en la planificación de proyectos de software.
  - Utilizar GIT como control de versiones
  - Utilizar Trac como gestor de tickets
  - Hacer uso de algún sistema de Integración Contínua.
  - Generar toda la documentación utlizando RestructuredText para perfecta integración con control de versiones.

Motivaciones personales

Si bien, existe un buen número de soluciones de mensajería instantánea y algunas de ellas soportan casi todas las características deseadas en el presente trabajo, las redes descentralizadas es un tema que me atrae profundamente, por lo que desarrollar un trabajo en este contexto, me parece una forma inmejorable de profundizar en el conocimiento de las mismas y me ofrece la oportunidad de aportar una pieza de sowftare operativa, basada en mis propios criterios y decisiones.

En concreto, la lectura de un artículo sobre la técnica "UDP Hole Punching" que permite comunicaciones directas entre equipos que, en principio, no son accesibles, me hizo intesarme en estas cuestiones y tras realizar algunos experimentos, me asaltó la inquietud de construir algo que "sirviera para algo" alrededor de dicha técnica.

Por otra parte, a lo largo de casi 20 años de experiencia profesional, la experiencia me dice que gran número de las empresas que se dedican al desarrollo de software están, aún lejos, de aplicar metodologías de ingeniería informática en sus proyectos.

Incluso empresas de renombre, empujadas casi siempre por los hitos y los *dead lines*, generan una enorme *deuda tecnológica* y el trabajo que realizan tiene, en demasiadas ocasiones, una carencia muy grande de calidad, tanto en diseño como en implementación.

Esto me motiva a realizar este trabajo en la forma en que me gustaría que mis proveedores lo realizaran.

Personalmente, también me apasiona aprender nuevas técnicas, herramientas y profundizar en cuestiones que tan sólo he podido rozar en mi experiencia profesional.

Elaboración de requisitos mediante Escenarios

Los escenarios describen situaciones del proceso del negocio, tanto del proceso observable actual como del proceso proyectado o futuro. En este último caso los escenarios resultan ser contenedores de la mayoría de los requisitos del sistema de software, pero no son los requisitos propiamente dichos.

"Muchas de las buenas prácticas o prácticas recomendadas en el proceso de desarrollo de software se basan en la existencia de un documento de requisitos en el que éstos se individualizan en forma precisa. En el presente artículo se presenta una estrategia para confeccionar un documento de requisitos a partir de escenarios ya construidos. La particularidad que presenta la misma es que los requisitos individualizados tienden a ser libres de conflictos y de ambigüedad a consecuencia del propio proceso de producción." [2]

Las situaciones observadas en el universo de discurso (UdeD) son la base para el conocimiento del problema y las situaciones deseadas son el bosquejo para los requisitos del sistema de software. Estas situaciones pueden representarse a través de escenarios. A los escenarios que plasman las situaciones observadas, los denominamos escenarios actuales. Escenarios futuros (EF) son aquéllos que modelan las situaciones proyectadas como solución a los problemas, demandas y necesidades planteadas en el UdeD y de acuerdo a los objetivos propuestos para el sistema. Estos EF incorporan muchos de los requisitos del software en sus descripciones, especialmente aquellos requisitos de naturaleza funcional.

Los EF son construidos desde el punto de vista del proceso del negocio, incorporando las acciones del actor "sistema de software". Es así, que el mundo descripto por estos escenarios no es observable sino que describen la forma en que se espera que se desenvuelva el negocio cuando se disponga del sistema de software que se planifica desarrollar. No toda la información incluida en los EF está directamente relacionada con el sistema de software, por el contrario muchos de ellos describen situaciones en las que no participa el sistema de software. Estas situaciones se constituyen en una eficaz descripción del contexto en el que el sistema se habrá de desenvolver. Naturalmente al describirse situaciones en las que el sistema de software realiza acciones ya sea en forma espontánea o como respuesta a estímulos de otro actor, se está hablando en forma implícita de las funcionalidades que deberá tener este sistema de software. En otras palabras, básicamente cada aceptación de un estímulo externo y cada respuesta del sistema está asociada con un requisito que se podría enunciar como "El sistema debe aceptar tal estímulo" o "El sistema debe proveer tal respuesta". [2]

El metamodelo de escenario

"El meta-modelo de escenario es una estructura compuesta por las siguientes entidades: título, objetivo, contexto, recursos, actores, episodios y excepciones, y el atributo restricción. Actores y recursos son dos componentes enumerativos. El título, el objetivo, el contexto y las excepciones son componentes declarativos, mientras que los episodios son un conjunto de sentencias en un lenguaje simple que dan una descripción operacional de comportamiento [...]" [2]*:*

Título: identificación del Escenario. En el caso de un subEscenario, el título es el mismo que la sentencia episodio, sin las restricciones.

Objetivo: finalidad a ser alcanzada. El Escenario describe la forma de lograr el objetivo.

Contexto: compuesto por al menos uno de los siguientes ítems:

- Ubicación Geográfica: lugar físico donde se produce el Escenario.
- Ubicación Temporal: especificación de tiempo para el desarrollo del Escenario.
- Precondición: estado inicial del Escenario.
- Recursos: elementos físicos o información relevantes que deben estar disponibles en el Escenario.
- Actores: personas, dispositivos o estructuras organizacionales que tienen un rol en el Escenario.
- Episodios: conjunto de acciones que detallan al Escenario y proveen su comportamiento.

Los episodios pueden ser de tres tipos: simples, condicionales u opcionales.

Los episodios simples son aquellos necesarios para concluir el desenvolvimiento del escenario.

Los episodios condicionales son aquellos cuya ocurrencia dependen de una condición específica. La condición puede ser interna o externa al escenario. Las condiciones internas pueden deberse a ubicaciones o precondiciones alternativas y a episodios previos.

Los episodios opcionales son aquellos que pueden o no ocurrir dependiendo de condiciones que no pueden ser explicitadas.

Independientemente del tipo, un episodio puede ser expresado como una acción simple o puede ser concebido en sí mismo como un escenario (denominado sub-escenario). Además, se pueden expresar episodios secuenciales y de orden indistinto.

Cada episodio puede llevar un atributo **Restricción**:

limitación o condición de calidad respecto a la realización del episodio, habitualmente estas restricciones se asocian a RNF (Requisitos No Funcionales).

Excepciones: generalmente reflejan la falta o mal funcionamiento de un recurso. Una excepción impide el cumplimiento del objetivo del Escenario. Se indica el tratamiento de la excepción a través de un Escenario (denominado Escenario Excepción) o de una acción simple.

Escenarios

Podemos distinguir dos tipos de escenarios claramente diferenciados:

- Escenarios de interacción, donde interviene un actor externo, sea un usuario o un proceso. Normalmente serán escenarios donde dicho actor interactua mediante la consola o api de más alto nivel.
- Escenarios de sistema, que son aquellos que de desarrollan internamente en el nodo sin que intervenga , ya, ningún elemento externo.

Escenarios de interacción

Escenario E.001. Instalación de Nodo

Si bien se podría pensar en una aplicación WEB, ejecutada sobre un pool de servidores, ZOE está pensado para ejecutarse en el equipo del usuario. No obstante, se podría usar fácilmente como core para aplicaciones WEB si así se deseara.

Lo primero que deberá hacer un usuario es hacerse con la instalación del software e instalarlo en su equipo.

Escenario E.001 Instalación de nodo		
Título		Instalación del nodo
Objetivo		Tener un nodo instalado en equipo propio
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	NA
	Precondición	Equipo compatible
	Recursos	Equipo, Instalador
	Actores	Usuario, Instalador
Episodios		
	El usuario se hace con el software	• eMail • Descarga directa • Soporte digital • Torrent • Otros
	El usuario ejecuta la instalación	Deben resolverse dependencias si es posible
	El instalador informa del resultado	

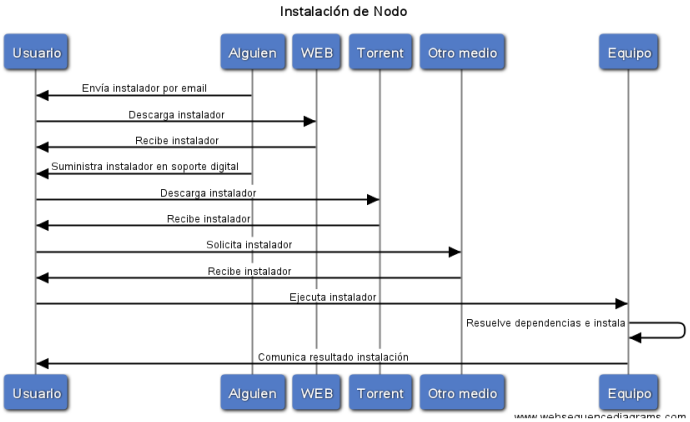


Fig. 1: Diagrama temporal instalación de nodo

Escenario E.002. Arranque de nodo

Una vez instalado el software, se procederá al arranque del mismo. Ya que no se dispone de servidor de registro y control de los ids de usuario, el usuario deberá proporcionar un uuid consistente en una dirección de correo electrónico que sea de su propiedad.

Esta dirección no se registrará en ningún otro nodo.

Escenario E.002 Arranque del Nodo		
Título		Arranque Nodo
Objetivo		Poner en marcha el nodo por primera vez
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	Arranque del nodo
	Precondición	Nodo instalado
	Recursos	Equipo, Nodo
	Actores	Usuario, Nodo, Presentador
Episodios		
	Arranque de los hilos de control	Core
		Node
		Net
		Contacts
		Console
		Storage
		Plugins

Escenario E.003. Consola Telnet

Dado que, en su modalidad GUIi, el la aplicación con dispondrá de entorno gráfico, toda interacción con el mismo deberá ser realizada, en principio, a través de su consola telnet.

Escenario E.003 Consola Telnet		
Título	Consola Telnet	
Objetivo	Acceder a la consola telnet del nodo	
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	NA
	Precondición	Nodo instalado
	Recursos	Equipo, Nodo, cliente telnet
	Actores	Usuario, Nodo
Episodios		
	El usuario hace telnet a IP:puerto de su nodo	Restricción: IP:Puerto debe ser accesible desde su ubicación
	El nodo responde con un mensaje de bienvenida y muestra la ayuda de comandos usables	

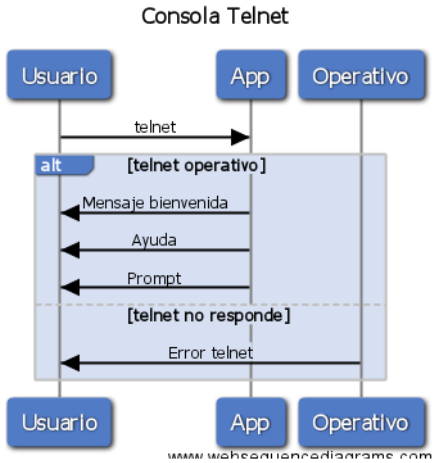


Fig. 2: Diagrama temporal de acceso a consola telnet

Escenario E.004. Funciones Consola

Escenario E.004 Funciones Consola		
Título	Funciones consola	
Objetivo	Funciones accesibles desde la consola telnet del nodo	
Contexto		
Ubicación geográfica	Dispositivo de usuario	
Ubicación temporal	NA	
Precondición	Nodo funcionando	
Recursos	Equipo, Nodo, cliente telnet	
Actores	Usuario, Nodo	
Episodios		
E.005 help	Muestra la ayuda de la consola	
E.006 login	Ejecuta login de usuario en la consola	
E.007 invite	Invitar a un contacto	
E.008 accept	Aceptar una invitación	
E.009 show contacts	Muestra contactos	
E.012 send	Enviar mensaje	
E.013 send crypted	Enviar mensaje encriptado	
E.014 history	Obtiene histórico de mensajes	
E.015 msgdel	Elimina mensaje	
E.016 python	Acceso a la consola python	
E.017 quit	Salir de la consola	
E.018 stop	Detener el nodo	

Escenario E.005. Help

Escenario E.005 Help		
Título		Help
Objetivo		Mostrar ayuda sobre los comandos disponibles en la consola
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	Cliente telnet conectado
	Precondición	Nodo funcionando
	Recursos	Equipo, Nodo, cliente telnet
	Actores	Usuario, Nodo
Episodios		
	El usuario escribe "help"	La consola muestra la lista de comandos disponibles e indicaciones sobre su itilización

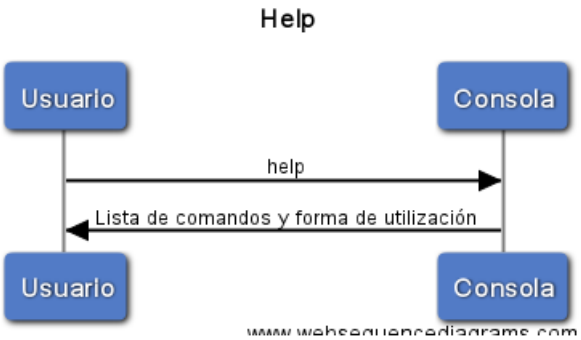


Fig. 3: Diagrama temporal de comando help

Escenario E.006. Login

Escenario E.006 Login		
Título	Login	
Objetivo	Autentificar usuario en la consola	
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	Consola telnet
	Precondición	Nodo jecutando
	Recursos	Equipo, Nodo, cliente telnet
	Actores	Usuario, Nodo
Episodios		
	El usuario ejecuta el comando login	RNF. password por defecto
	Si es correcto	Mensaje OK, El usuario puedo utilizar todas las funciones de la consola E.004
	Si no es correcto	Mensaje Error. Se muestra ayuda de nuevo
		RNF. Aún no logado, el nodo ejecuta toda su lógica interna enviando y recibiendo mensajes pendientes

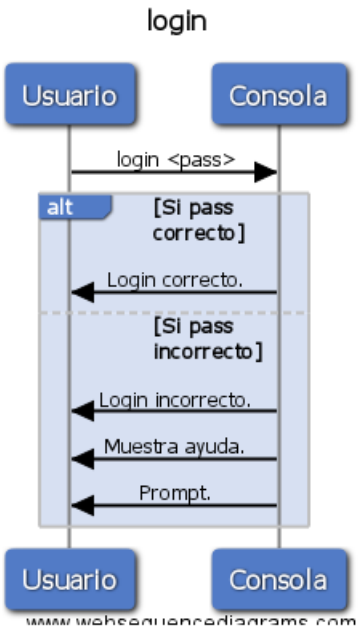


Fig. 4: Diagrama temporal de comando login

Escenario E.007. Invite

Escenario E.007 Invite		
Título		Invite
Objetivo		Invitar a un usuario a ser contacto
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	Nodo ejecutandose
	Precondición	Nodo activado
	Recursos	Equipo, Nodo, cliente telnet
	Actores	Usuario, Nodo
Episodios		
	El usuario ejecuta el comando invite con email de contacto	RNF. Se ejecuta el escenario interno invitación de contacto E.024
	Si el contacto acepta	Se registra el remoto como contacto
	Si el contacto no hace nada	La invitación sigue como pendiente
	Si el contacto rechaza	La invitación sigue como pendiente
		RNF. Una invitación queda como pendiente hasta que: • El remoto la acepta • El usuario la cancela

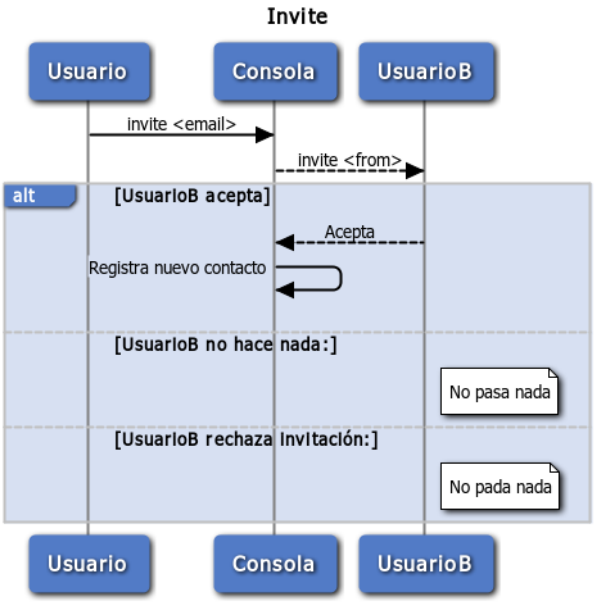


Fig. 5: Diagrama temporal de comando invite

Escenario E.008. Accept

Escenario E.008 Accept		
Título		Accept
Objetivo		Aceptar una invitación de contacto
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	Nodo ejecutandose
	Precondición	Invitación recibida
	Recursos	Equipo, Nodo, cliente telnet
	Actores	Usuario, Nodo
Episodios		
	El usuario obtiene la invitación pendiente	
	El usuario ejecuta accept para la invitación pendiente	Ambos usuarios son contactos • RNF. Se pueden aceptar varias invicationes a la vez • RNF. Se ejecuta internamente E.024 Aceptación de contacto
	Si el contacto no hace nada	No se hace nada. La invitación sigue como pendiente
	Si el contacto rechaza	La invitación desaparece como pendiente RNF. No se envía nada al remoto
		RNF. Una invitación recibida queda como pendiente hasta que: • Se acepta • Se elimina

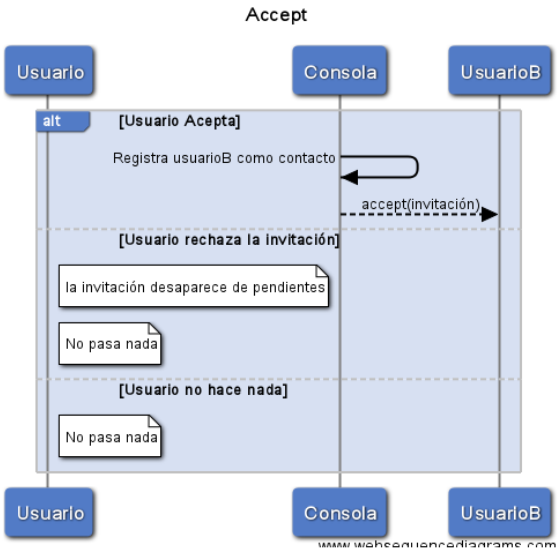


Fig. 6: Diagrama temporal de comando accept

Escenario E.009. Show Contacts

Escenario E.009 Show Contacts		
Título		Show contacts
Objetivo		Muestra información de contactos
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	Nodo ejecutandose
	Precondición	NA
	Recursos	Equipo, Nodo, cliente telnet
	Actores	Usuario, Nodo
Episodios		
	El usuario solicita información de contactos	El sistema retorna la información solictada o mensaje de error

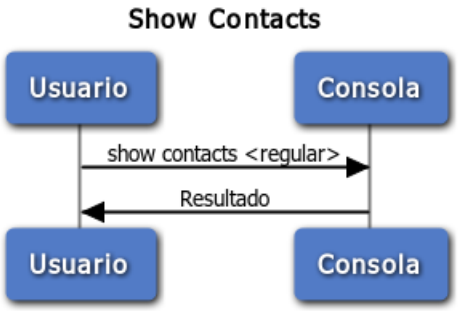


Fig. 7: Diagrama temporal de show contacts info

Escenario E.012. Send

Escenario E.012 Send		
Título		Send
Objetivo		Enviar un mensaje de usuario
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	Nodo ejecutandose
	Precondición	Contacto aceptado
	Recursos	Equipo, Nodo, cliente telnet
	Actores	Usuario, Nodo
Episodios		
	El usuario ejecuta send a un contacto	• RNF. Se encola el mensaje a enviar • Condición: El destino debe ser contacto aceptado
	El sistema notifica resultado	• RNF. Resultado de operación

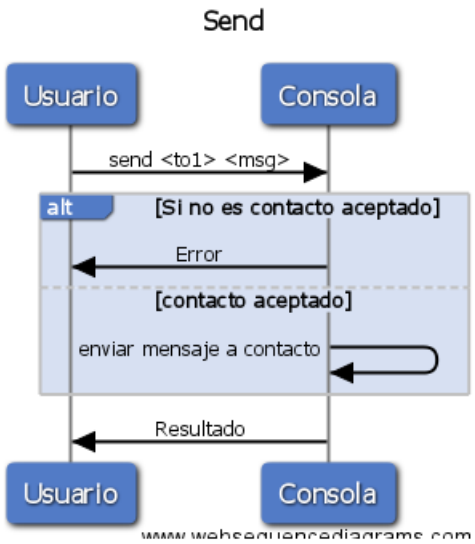


Fig. 8: Diagrama temporal de comando send

Escenario E.013. Send Crypted

Escenario E.013 Send Crypted		
Título		Send Crypted
Objetivo		Enviar un mensaje encriptado
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	Nodo ejecutandose
	Precondición	Contacto aceptado Clave publica de contacto
	Recursos	Equipo, Nodo, cliente telnet
	Actores	Usuario, Nodo
Episodios		
	El usuario ejecuta send encriptado a un contacto	• Condición: el destino debe ser contacto aceptado • Se ecripta el mensaje con la clave publica del contacto • Se firma el mensaje con la clave privada del sender • RNF. A nivel interno no hay ninguna diferencia en enviar un mensaje encriptado o no. Será internamente mediante un flag donde se aplicará la encriptación. • RNF El usuario que envía debe tener la clave pública del destinatario. • RNF El destinatario debe tener la clave pública del que envía.
	El sistema notifica resultado	• RNF. Resultado de operación

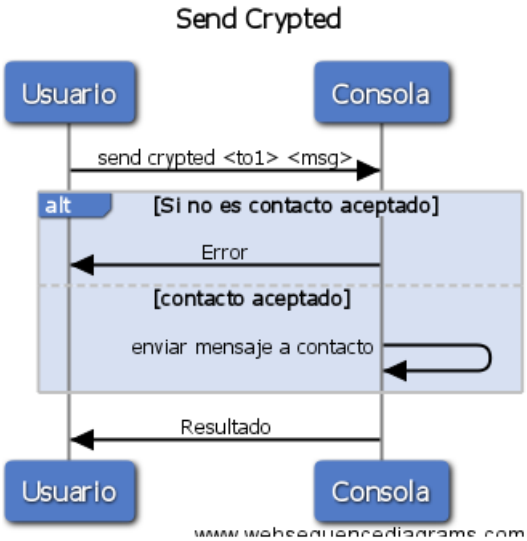


Fig. 9: Diagrama temporal de comando send crypted

Escenario E.014. History

Escenario E.014 History		
Título		History
Objetivo		Obtiene histórico de mensajes
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	Nodo ejecutandose
	Precondición	N/A
	Recursos	Equipo, Nodo, cliente telnet
	Actores	Usuario, Nodo
Episodios		
	El usuario ejecuta history	
	El sistema retorna resultado	

Fig. 10: Diagrama temporal de comando history

Escenario E.015. msgdel

Escenario E.015 msgdel		
Título		msgdel
Objetivo		Eliminar un mensaje
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	Nodo ejecutandose
	Precondición	Mensaje recibido
	Recursos	Equipo, Nodo, cliente telnet
	Actores	Usuario, Nodo
Episodios		
	El usuario ejecuta msgdel y el id de mensaje	
	El sistema retorna resultado	

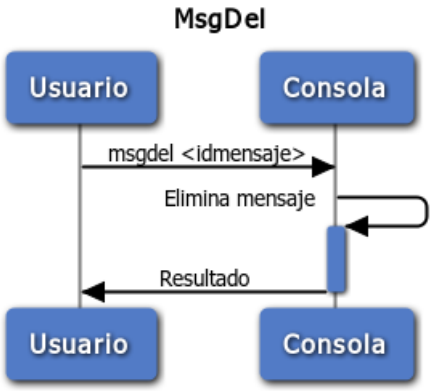


Fig. 11: Diagrama temporal de comando msgdel

Escenario E.016. python

Escenario E.016 python		
Título		python
Objetivo		Dar acceso a la consola de python en el espacio de ejecución
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	Nodo ejecutandose
	Precondición	NA
	Recursos	Equipo, Nodo, cliente telnet
	Actores	Usuario, Nodo
Episodios		
	El usuario ejecuta python	
	El sistema retorna prompt de python	RNF. controlar modulos potencialmente dañiños

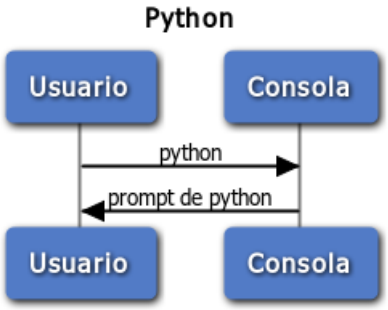


Fig. 12: Diagrama temporal de comando python

Escenario E.017. quit

Escenario E.017 quit		
Título	quit	
Objetivo	Abandonar el entorno actual	
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	Nodo ejecutandose
	Precondición	NA
	Recursos	Equipo, Nodo, cliente telnet
	Actores	Usuario, Nodo
Episodios		
	El usuario ejecuta quit	
	Si se está en consola python Si se está en consola comandos	Se retorna a consola comandos Se abandona consola telnet

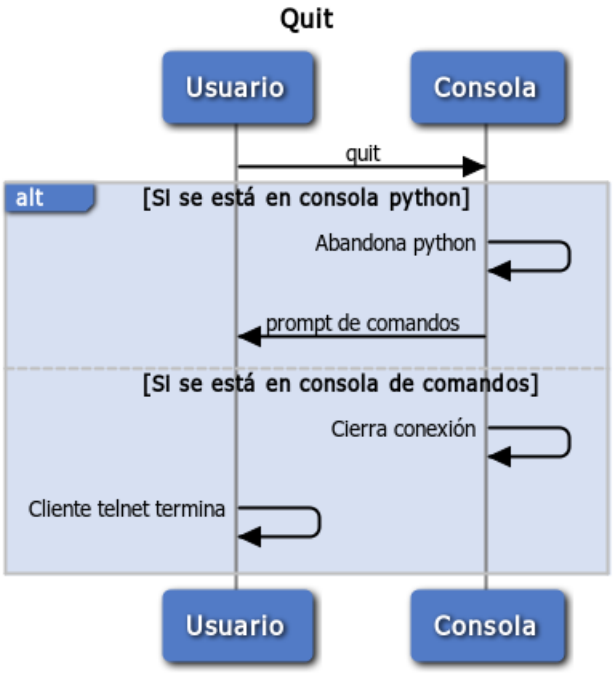


Fig. 13: Diagrama temporal de comando quit

Escenario E.018. stop

Escenario E.018 stop		
Título		stop
Objetivo		Detiene el nodo
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	Nodo ejecutandose
	Precondición	NA
	Recursos	Equipo, Nodo, cliente telnet
	Actores	Usuario, Nodo
Episodios		
	El usuario ejecuta stop	
	Tras confirmación, el nodo se detiene	

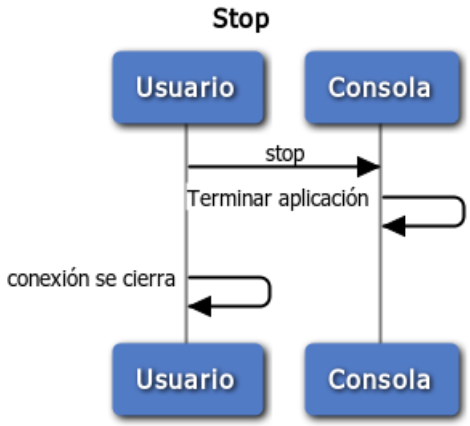


Fig. 14: Diagrama temporal de comando stop

Escenarios de Sistema

Escenario E.019. Arranque de Nodo

Un nodo, al arrancar, se publicará automáticamente en sus Presentadores (nodos "fijos"). En dicha publicación el nodo envía periódicamente información sobre su uuid (hash de la dirección de correo) e IP:puerto locales. La IP y puerto que expone a internet la obtiene el propio Presentador a partir del datagrama recibido.

Escenario E.019 Arranque de nodo		
Título		Arranque de Nodo
Objetivo		Conectar nodo a la red
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	NA
	Precondición	Nodo instalado
	Recursos	Equipo, Nodo
	Actores	Nodo, Presentador
Episodios		
	El nodo publica sus datos en el Presentador	RNF. El nodo no vuelve a enviar email
		RNF. El nodo envía con cierta frecuencia un paquete UDP con información sobre su localización, IP publica, IP interna y puerto interno, así como otros datos que puedan ser de utilidad.

Fig. 15: Diagrama temporal de arranque de nodo

Escenario E.020. Invitación de contacto

Lo primero que debe hacer un usuario para comunicarse con otro es realizar una invitación de contacto. ZOE no está pensado para que todos hablen con todos, sino para establecer una red de amigos a amigos F2F.

Lo único que debe hacer un usuario para invitar a otro es enviar un mensaje de invitación al otro contacto.

Escenario E.020 Envío Invitación de contacto		
Título		Envío Invitación de contacto
Objetivo		Invitar a contacto
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	NA
	Precondición	Nodo instalado y operativo
	Recursos	NodoA, NodoB
	Actores	NodoA, NodoB
Episodios		
	A localiza a B A envía invitación a B	RNF. Uso del presentador E.007

Fig. 16: Diagrama temporal de Invitación de contacto

Escenario E.021. Envío de mensaje

Una vez que dos nodos han sido "conectados", el envío de mensajes se produce directamente entre ambos, sin que, en principio, intervenga ningún nodo intermedio. Aún así, la comunicación entre ambos es encriptada, opcionalmente, mediante clave pública-clave privada.

> ***Note***
>
> Obviamente, dos nodos que quieran comunicarse de manera encriptada, deberán intercambiar sus claves públicas. Puesto que un canal no seguro no es seguro hasta que lo sea, la mejor forma de intercambiar las claves sería "en mano", pero dependiendo del nivel de paranoia de los usuarios, se podrían intercambiar por correo electrónico, recursos compartidos en la nube, etc ...

Se establecen dos categorías de mensajes:

- Mensajes sin garantía de entrega, al más puro estilo UDP. El mensaje se envía y no se espera confirmación.
- Mensajes con garantía de entrega: el envío del mensaje se reintenta hasta que bien:
  - El mensaje haya sido entregado
  - El usuario cancele el mismo

Sólo los mensajes con *payload*, generados por el usuario, y los correspondientes a invitaciones o aceptación de invitaciones, se envían con garantía.

Escenario E.021 Envío de mensaje		
Título		Envío de mensaje
Objetivo		Enviar un mensaje
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	Nuevo mensaje a enviar
	Precondición	Nodo instalado y activado
	Recursos	NodoA, Presentador, NodoB
	Actores	Nodo A, NodoB, presentador
Episodios		
	Si envío con garantía	E.027 Envío de mensaje con garantía
	Si envío sin garantía	E.026 Envío de mensaje sin garantía
	A localiza a B	RNF. Uso del presentador E.007

Escenario E.022. Envío de mensaje sin garantía

Los mensajes sin garantía se envían una sóla vez y no esperan ningún tipo de *ack* por parte del destinatario.

Escenario E.022 Envío de mensaje sin garantía		
Título		Envío de mensaje sin garantía
Objetivo		Enviar un mensaje sin necesidad de garantía de entrega
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	Nuevo mensaje a enviar sin garantía
	Precondición	Nodo instalado y activado
	Recursos	NodoA, NodoB
	Actores	Core, Storage, Net, Presentador
Episodios		
	Si requiere historia	E.010 Storage Almacena mensaje
	Si no requier historia	No se almacena el mensaje
	A localiza a B	RNF. Operativa del presentador E.032
	Si B localizado	E.026 Despachar mensaje
	Si B no localizado	Si mensaje registrado, Storage marca como no enviado

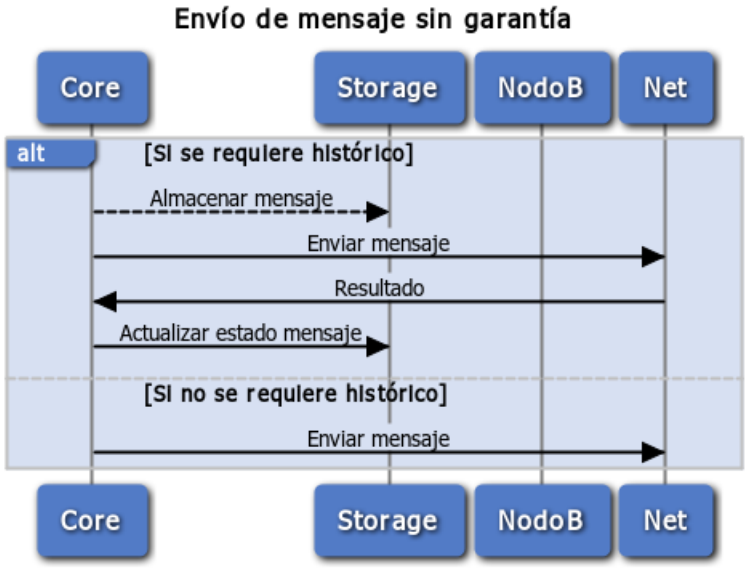


Fig. 17: Diagrama temporal de envío de mensaje sin garantía

Escenario E.023. Envío de mensaje con garantía

Los mensajes de garantía se intentarán entregar hasta que se den una de estas tres circunstancias:

- Que el mensaje haya sido entregado
- Que el usuario lo elimine.

Además, los mensajes con garantía siempre se registran en la base de datos local.

Escenario E.023 Envío de mensaje con garantía		
Título		Envío de mensaje con garantía
Objetivo		Enviar un mensaje con necesidad de garantía de entrega
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	Nuevo mensaje a enviar con garantía
	Precondición	Nodo instalado y operativo
	Recursos	Nodo
	Actores	Mensaje, Core, Storage, Net, Presentador
Episodios		
	Almacenar mensaje	Storage almacena mensaje
	A localiza a B	RNF. Uso del presentador E.007
		RNF. En ningún momento vuelve a viajar el email de B
	Si B localizado	E.026 Net despacha mensaje
	Si B no localizado	Dejar mensaje como pendiente
		RNF. En este caso el mensaje queda en el almacenamiento como pendiente de entregar y se seguirá intentando hasta que: • B reciba el mensaje • A cancele el mensaje

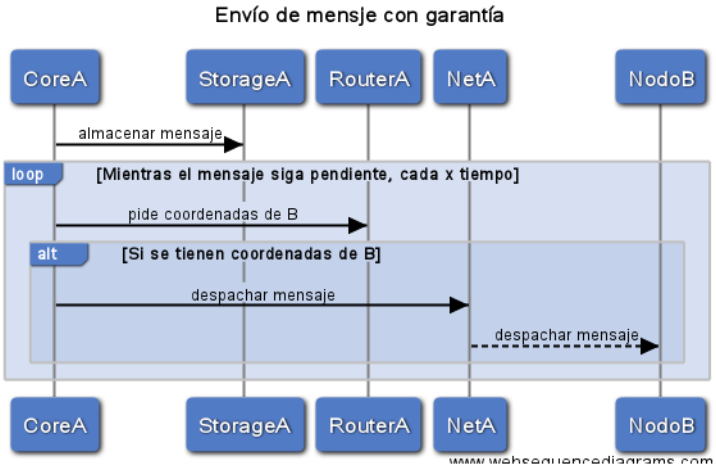


Fig. 18: Diagrama temporal de envío de mensaje con garantía

Escenario E.024. Aceptación de contacto

Cuando un nodo recibe una invitación de contacto el usuario deberá decidir qué hace con dicha invitación: Si aceptarla o eliminar el contacto invitante.

- Si la acepta, enviará la aceptación a A y ambos nodos serán "amigos".
- Si elimina el contacto invitante, su nodo no enviará ningún mensaje al invitador y para este la invitación estará pendiente de aceptar

Escenario E.024 Aceptación de contacto		
Título		Aceptación de contacto
Objetivo		Aceptar un contacto del que se ha recibido invitación
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	B recibe invitación de A
	Precondición	Invitación pendiente de aceptar
	Recursos	NodoA, NodoB
	Actores	NodoA, NodoB
Episodios		
	Si el usuario B acepta	El nodo de B envía la aceptación a A
	Si B no acepta	RNF Si no hace nada, el nodo no hace nada. La invitación queda pendiente RNF Si elimina la invitación, no se envía ningún mensaje. Le quedara a A como pendiente de aceptar por B

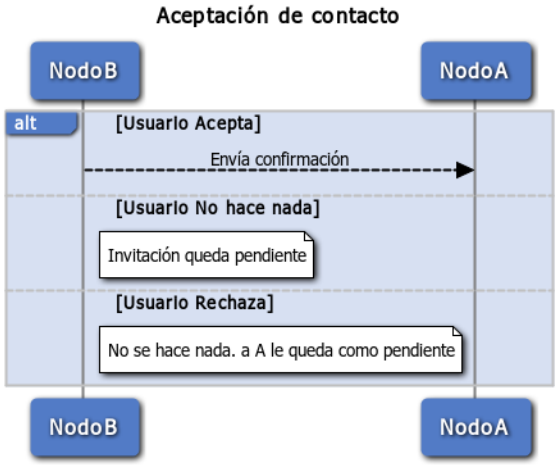


Fig. 19: Diagrama temporal de Aceptación de Contacto

Escenario E.025. Almacenar Mensaje

El sistema requiere persistencia de los mensajes enviados y recibidos. En futuras implementaciones, este almacenamiento permitirá consultar históricos, cancelar envíos pendientes, aceptar invitaciones pendientes recibidas o cancelar invitaciones pendientes enviadas, entre otras muchas cosas.

Escenario E.025 Almacenar Mensaje		
Título		Almacenar Mensaje
Objetivo		Registrar un mensaje en almacenamiento local
Contexto		
	Ubicación geográfica	Nodo de A
	Ubicación temporal	Nodo operativo y operativo
	Precondición	Nodo instalado y operativo
	Recursos	Nodo A, mensaje
	Actores	Nodo A
Episodios		
	Completar metadatos	• id de mensaje. RNF. delegado en motor DB • to: email de destinatario • tipo de mensaje: DATA, CAL, HEL ... • timestamp: UTC timestamp • payload: payload • ackd: entrega confirmada • encrypt: si el mensaje ha de ser encriptado
	Pedir a storage que lo almacene	• RNF. Independiente de motor de DB

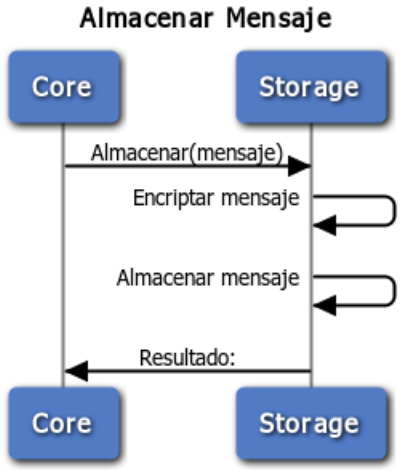


Fig. 20: Diagrama temporal de Almacenar Mensaje

Escenario E.026. Despachar mensaje

El despachador de mensajes es un simple componente que hace de intermediario entre el módulo de almacenamiento y el módulo de red.

Escenario E.026 Despachar Mensaje		
Título		Despachar Mensaje
Objetivo		Enviar mensaje a B utilizando la capa de red
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	Mensaje pendiente de enviar
	Precondición	Existe un mensaje pendiente de enviar a B
	Recursos	Nodo A
	Actores	Nodo A, NodoB
Episodios		
	Se le pide a la capa de red que lo envíe	Envío por red RNF. No se espera respuesta del remoto
	La capa de red retorna resultado	RNF. Sólo de la operación de enviar en sí misma, no de si el remoto recibió o no el mensaje.

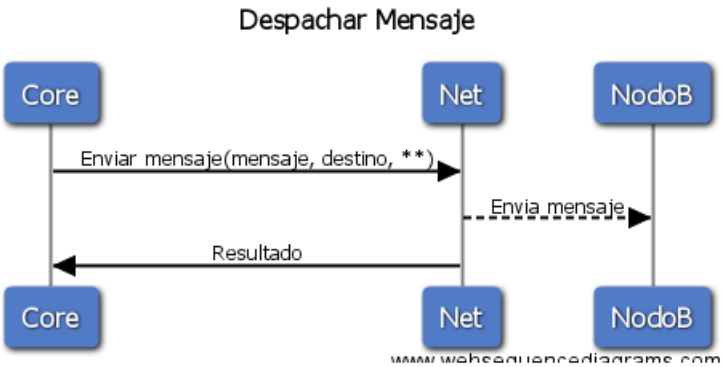


Fig. 21: Diagrama temporal de Despachar mensaje

Escenario E.027. Obtener Mensajes Pendiente

En este escenario, tanto el despachador de mensajes, como otros procesos del sistema internos o externos solicitan mensajes pendientes al módulo de almacenamiento.

Escenario E.027 Obtener Mensajes Pendientes		
Título		Obtener Mensaje Pendiente
Objetivo		Obtener de la capa de storage el siguiente mensaje pendiente
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	Mensaje pendiente de enviar
	Precondición	Existe un mensaje pendiente de enviar a B
	Recursos	Nodo A
	Actores	Nodo A, despachador, otros
Episodios		
	Storage retorna la lista de mensaje s pendientes	

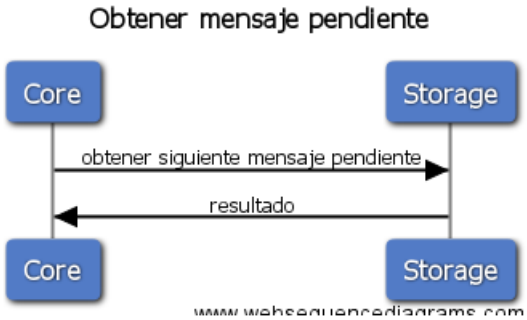


Fig. 22: Diagrama temporal de obtener mensaje pendiente

Escenario E.028. Envío por Red

El módulo de red se encarga de encargar al protocolo adecuado que codifique el mensaje y lo envía por el canal correspondiente.

Escenario E.028 Envío por Red		
Título		Envío por Red
Objetivo		Enviar mediante la capa de red un mensaje
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	Mensaje encolado para enviar
	Precondición	Existe un mensaje pendiente de enviar a B
	Recursos	CoreA, NetA, RouterA
	Actores	Nodo A, Nodo B
Episodios		
	Net codifica el mensaje con protocolo ZOE	E.014 Codificación ZOE
	Net pide a Routing la dirección conocida de B	E.015 Query Address
	Net envía el mensaje	RNF. Desantendido: No se espera respuesta

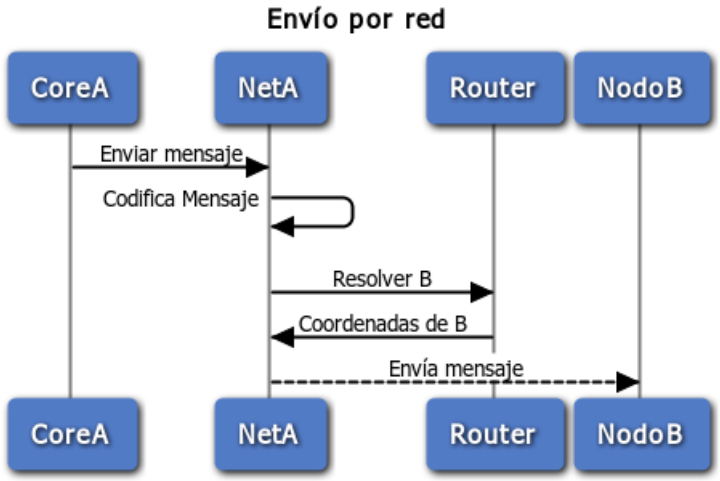


Fig. 23: Diagrama temporal de envío por red

Escenario E.029. Codificación ZOE

El protocolo estándar de ZOE será pZoe. Todos los mensajes, antes de ser enviados por el módulo de red deberán ser formateados según este protocolo, al igual que todos los mensajes entrantes deberán ser parseados por el mismo.

Escenario E.029 Codificación ZOE		
Título		Codificación ZOE
Objetivo		Construir un mensaje en protocolo ZOE
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	Mensaje encolado en Red para enviar
	Precondición	Existe un mensaje pendiente de enviar a B
	Recursos	Nodo A
	Actores	Nodo A
Episodios		
	Net solicita al proto ZOE que construya un mensaje válido	RNF. Definición proto ZOE
	proto ZOE retorna el mensaje construido	

Todo mensaje de ZOE está compuesto por los siguientes campos:

Campo	Tipo	Descripción
cmd	text	Comando
mid	uuid	Identificador único de mensaje
from	hash	Identificador de remitente
to	hash	Identificador de destino
msg_type	text	Tipo del mensaje
payload	text	Payload del mensaje
encrypt	bool	Indica si el mensaje ha de ser encriptado

La codificación utilizada, como ejemplo, es una sencilla codificación cpickle [1].

Zoe maneja un conjunto reducido de tipos de mensajes que se muestran a continuación:

Tipos de mensajes:

Tipo	Descripción
HEL	Publicación de datos en nodo remoto
SEA	Preguntar a nodos conocidos por otro nodo
DATA	Mensaje con "payload"
CAL	Solicitud de "llamada" a otro nodo
ACK	ack de mensaje recibido

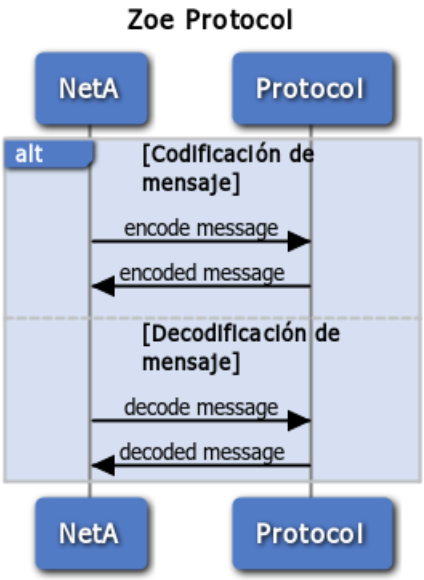


Fig. 24: Diagrama temporal de codificación y decodificación en Protocol

Escenario E.030. Query Address

Cuando un nodo quiera comunicarse con otro, lo primero que ha de hacer es localizar su dirección y puerto. Si el nodo remoto ha sido "visto" recientemente, dicha dirección y puerto estará en su memoria local.

Si el nodo no está en su lista de nodos "activos", el módulo de enrutamiento comenzará un proceso de punching [#punching]_.

Escenario E.030 Query Address		
Título		Query Address
Objetivo		Obtener la dirección de un remoto
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	Mensaje encolado en Red para enviar
	Precondición	Existe un mensaje pendiente de enviar a B
	Recursos	Nodo A, Router
	Actores	Nodo A, B, Router
Episodios		
	Router comprueba si conoce la ubicación de B	RNF. Lista local
	Si la conoce	Retorna la IP:puerto de B
	Si no la conoce	• Router retorna fallo (None) • A envia SEARCH (SEA) a cada nodo conocido para B

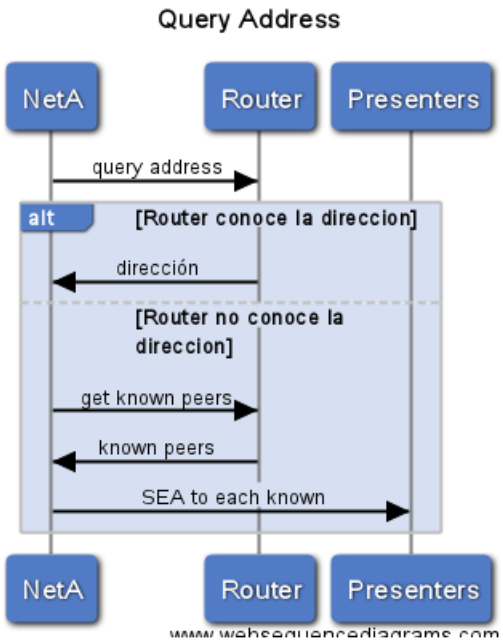


Fig. 25: Diagrama temporal de query address

Escenario E.031. Discover

Mediante el uso de un Presentador, que tiene su puerto UDP accesible, los nodos pueden solicitar conexión con otros. Una vez que ambos nodos conocen la dirección del contrario, se puede establecer una conexión P2P entre ellos, aunque sus puertos UDP no sean accesibles. Esta cuestión será analizada en profundidad posteriormente.

Escenario E.031 Discover		
Título		Discover
Objetivo		Obtener conexion P2P con remoto
Contexto		
	Ubicación geográfica	Dispositivo de usuario
	Ubicación temporal	Mensaje encolado en Red para enviar
	Precondición	Existe un mensaje pendiente de enviar a B
	Recursos	Nodo A, Presentador
	Actores	Nodo A, B, Presentador
Episodios		
	Router envía un mensaje	RNF. Este mensaje se repite periódicamente hasta que:
	Si se recibe mensaje de Presentador sobre ubicación de B	• Se obtiene dirección de B • No hay nada pendiente para B Se envía mensaje HEL a B
	Si A recibe mensaje de B	Se registra su dirección actual
Restricción		
	Operativa de Presentador	E.032

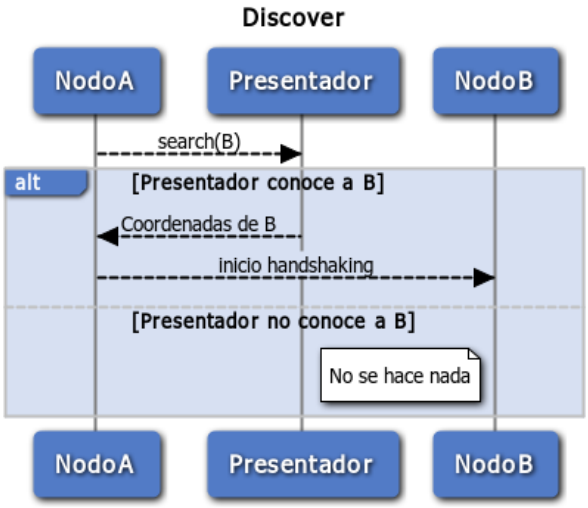


Fig. 26: Diagrama temporal de discover

Escenario E.032. Operativa Presentador

Todos los nodos son iguales. Lo único que diferencia a un presentador es que su puerto UDP es accesible y que suele estar "vivo".

El Presentador no tiene almacenamiento. Sus función exclusiva es:

- Poner en contacto dos nodos que quieren conectarse

Escenario E.032 Operativa Presentador		
Título		Operativa Presentador
Objetivo		Servir de directorio para localización entre nodos
Contexto		
	Ubicación geográfica	Nodo estable
	Ubicación temporal	Cuando se quieren conectar dos nodos
	Precondición	Puerto UDP accesible
	Recursos	Nodo A, Presentador, Nodo B
	Actores	Nodo A, Presentador, Nodo B
Episodios		
	Los nodos se publican	Cada nodo envía, con una latencia fija, un mensaje de publicación a los nodos conocidos, entre ellos a los presentadores. En este mensaje se envía información referente al nodo como: • uuid del nodo • ip:puerto local • ... RNF. Los nodos, incluidos los presentadores, no tienen memoria sobre direccionamiento. Los nodos registrados, que lo están sólo en memoria, se eliminan cada cieto tiempo.
	P recibe un mensaje SEA(B)	Mediante este mensaje, A está solicitando al Presentador información sobre dónde se encuentra B
	Si P conoce a B envía a A las coordenadas de B y a B las coordenadas de A	RNF. P envía sendos mensajes CALL(A) y CALL(B) a ambos nodos Estos mensajes incorporan información sobre direcciones privadas.
	Si P no conoce a B no hace nada	

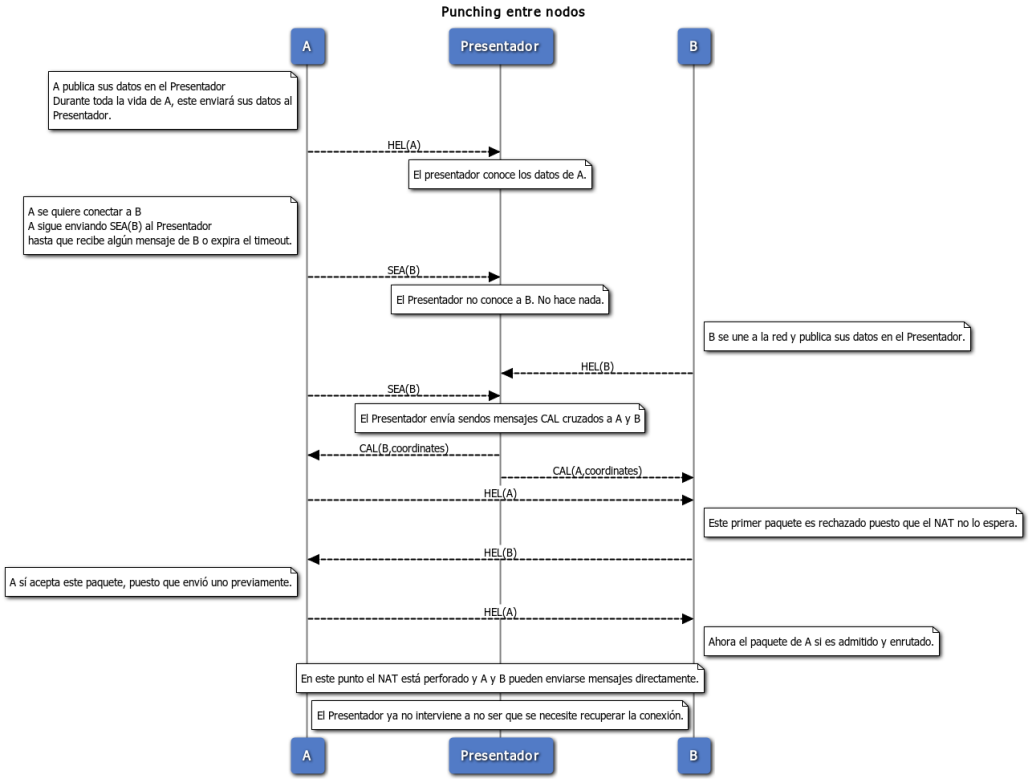


Fig. 27: Diagrama temporal simplificado de Operativa Presentador

Implementación

Entorno de desarrollo

En una empresa de desarrollo de software, normalmente las herramientas utilizadas, el entorno de desarrollo y la metodología aplicada están -o deberían estar- bien definidas y no supone una tarea adicional que deberá planificarse junto con las demás.

En este trabajo, sin embargo, no están definidas ni las herramientas ni el entorno de desarrollo ni la metodología.

Esto constituirá, por tanto, una fase más del proyecto, que deberá ser resuelta con esmero puesto que cambiar a mitad de proyecto de herramientas, entorno de trabajo o metodología normalmente producirá un gran trastorno, si es que fuera posible.

Lo normal si se producen errores en la resolución de estos aspectos, es que se sigan "sufriendo" hasta el final del proyecto sin ser modificados, puesto que el coste en tiempo y dinero no sería asumible.

Herramientas:

- Sistema Operativo: Linux Debian 6 y Mint 12
- Lenguaje de programación: Python2.6 y bash
- IDE: Wing 4.0 y vim
- Documentación: RestructuredText
- Motor de bases de datos: sqlite3
- Expresiones regulares: Kodos
- Control de versiones: Git, gitk y git gui
- Gestor de tickets: Trac
- Backup adicional y sincronización entre equipos: Dropbox
- Generación de gráficos: GnuPlot
- Tratamiento de imágenes: Gimp
- Diagramas UML: plantuml, websequencediagrams y dia
- Tests unitarios: pytest
- Ajuste a PEP8: pylint
- Integración continua: Jenkins
- make

A continuación se hará una breve justificación de cada una de las herramientas utilizadas. Lo cierto es que el análisis de cada una de ellas podría constituir, por si mismo, tema para un Proyecto de Fin de Carrera.

Sistema Operativo

Los sistemas operativos utilizados en el desarrollo han sido Debian7 sobre un DELL 260ST y Mint 12 sobre un portátil Sony VAIO.

La elección de Linux se debe a que entiendo que un sistema UNIX es el sistema idóneo para soportar desarrollos de software como el realizado en este trabajo. No es del alcance de este trabajo entrar en un debate sobre qué sistema operativo es mejor o peor, pero la experiencia al respecto es contundente.

Incluso si se tiene que desarrollar, por imperativos del proyecto, sobre otros operativos, Linux resulta perfecto para integrar todas las herramientas necesarias.

Lenguaje de programación

Este trabajo consiste en la realización de un software en el que la clave está en comunicaciones entre ordenadores o dispositivos a través de sockets UDP, traspasando NATs y cortafuegos.

Además, tiene un pequeño soporte de bases de datos para hacer posible su funcionamiento.

Con estas premisas, cualquier lenguaje existente que tenga soporte para sockets UDP y cualquier forma de almacenamiento, sería susceptible de ser utilizado.

Algunos de los lenguajes más populares que podrían haberse utilizado son:

- C++ en cualquiera de sus sabores,
- Java
- Visual Basic
- PHP
- etc

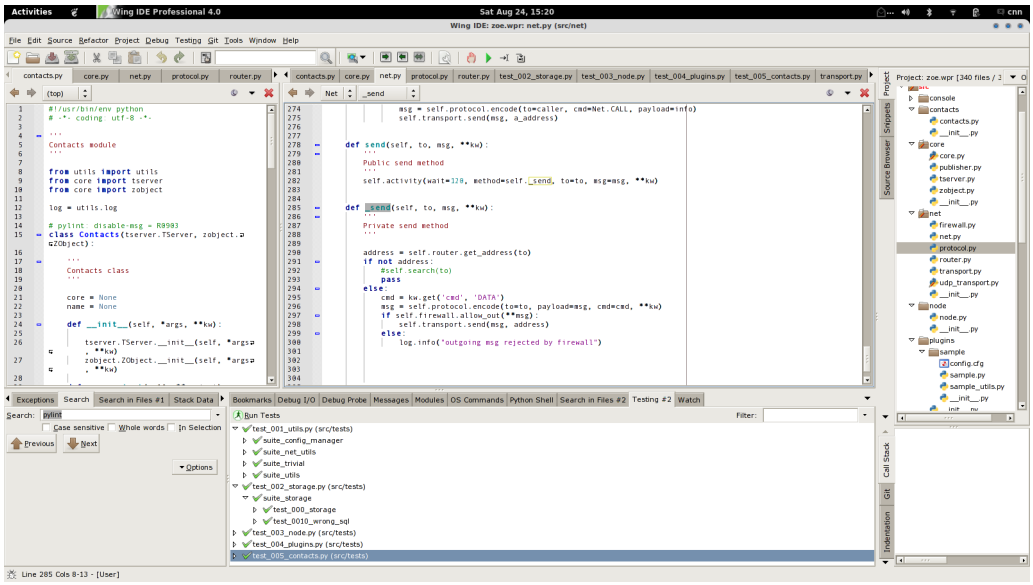
Sin embargo, se ha dedicido utilizar Python, por tratarse de un lenguaje elegante, estructurado, limpio, multiplataforma y de muy ràpido desarrollo (se dice que una misma rutina en java ocupa entre 5 y 10 veces más que la misma en python ...)

C++, por supuesto, es más rápido que python. También java lo es en muchas ocasiones. Sin embargo, la velocidad de ejecución que ofrece python es más que suficiente para cubrir las necesidades de este trabajo.

En cualquier caso, no se hacen uso de funcionalizades exclusivas de python, como funciones lambda [27] o yields [28], de manera que el *"porting"* a cualquier otro lenguaje podría ser directo.

IDE

WingIDE (http://wingware.com/?gclid=COPx_7WVlrkCFe_LtAodnwoAZw) es un IDE muy ligero y específico para python. Si bien no es libre, existe una versión gratuita perfectamente utilizable.



La elección de WingIDE se debe a ser el normalmente utilizado por el autor de este trabajo.

Obviamente, cualquier IDE -preferiblemente con capacidad de depuración sobre python - hubiera sido utilizable para la realización del software, como el popular Eclipse [3].

VI

VI es un editor de texto disponible probablemente en cualquier instalación de cualquier sistema operativo tipo UNIX. Al no depender de un entorno gráfico, vi se puede utilizar para realizar cambios en archivos locales o remotos (via ssh, por ejemplo) con enorme rapidez.

Restructured Text

Diseñado originalmente para extraer de forma automática documentación de fuentes en python, RestructuredText proporiona una manera muy cómoda de poder generar documentos en un buen número de formatos como html, docbook, html, LaTeX o pdf.

Los fuentes de los documentos rst son ficheros ascii planos, que hacen que se integren perfectamente en sistemas de control de versiones y de cambios y que las mezclas se puedan realizar perfectamente siempre que sea posible, como si se estuviera mezclando código fuente.

Una clara ventaja sobre otras alternativas similares como docbook o LaTeX es que los fuentes de RST son muy limpios y se pueden leer perfectamente en ASCI

Sqlite3

Todos los accesos a datos almacenados se realizan a través de un interface Storage. Esto tiene la evidente ventaja de que podrá utilizarse cualquier clase que implemente ese interfaz.

Puesto que el almacenamiento estará completamente desacoplado de la lógica de la aplicación en un patrón tipo MVC, el almacenaje podría ser cualquier motor de bases de datos, file system, S3, o cualquier otro que pudiera imaginarse o diseñarse.

En este trabajo se ha utilizado Sqlite3 por ser un motor de base de datos muy ligero y que no requiere de servidor alguno.

Kodos

Una de las formas más potentes de procesar cadenas de texto es la utilización de expresiones regulares. Kodos (http://kodos.sourceforge.net/home.html) es una herramienta específica para generar y probar expresiones regulares de python

GIT

Actualmente resulta impensable el desarrollo de software sin utilizar un sistema de control de versiones.

A parte de soluciones comerciales, algunas de las más populares son CSV o SVN (Subversion). Como sistemas de control de versiones más modernos destacan Mercurial o Git, siendo éste último el escogido para este trabajo ya que es extremadamente rápido y fiable.

Si bien la curva de aprendizaje de GIT es algo más complicada que csv o svn, las ventajas que reporta son grandes puesto que gestiona perfectamente el control de diferentes ramas y aportaciones a cada una de ellas. Además es muy rápido y ligero y resuelve las mezclas con gran eficacia.

Trac

Al igual que no se puede concebir un desarrollo de software sin control de versiones, tampoco se puede hacer sin control de tareas.

- ¿ Qué tareas hay pendientes ?
- ¿ En cuales se está trabajando ?
- ¿ Quién es responsable de cada tarea ?
- ¿ Cuanto tiempo va a consumir o ha consumido ?
- ¿ Qué subtareas (breakdown) tiene una tarea ?
- ¿ Qué dependencia existe entre tareas y cuales son bloqueantes ?
- etc

Existen muchos gestores de tickets, siendo en el software libre uno de los más populares Trac, puesto que es altamente configurable y dispone de multitud de plugins para integrarse con otros sistemas.

El Departamento de Lenguajes y Sistemas informáticos proporciona un espacio de Trac a los alumnos de Proyecto de Fin de Carrera.

Si bien se trata de un trac escueto, proporciona una funcionalidad básica de gestión de tickts que sirve para ilustrar su funcionamiento.

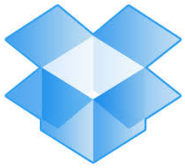
http://1984.lsi.us.es/pfe/trac/pfe-f2f-im

En todo caso, en un trac propio, se puede configurar para que sea enormente más útil, con breakdown de tareas, bloqueos, recursos, integración con git, etc ...

Backup

Puesto que todo el proyecto está en GitHUB, este supone un backup de hecho. Sólo con hacer un git clone de la URL apropiada se tendrá una copia del proyecto en local.

Como backup adicional, el directorio del proyecto y todos los que cuelgan de él, se mantienen sincronizados en Dropbox.



Una manera nada sofisticada y muy efectiva de tener backup de un proyecto.

Gráficos basados en datos

Los gráficos basados en datos utilizados en este proyecto se definen en ascii, lo cual permite automatizarlos, si fuera necesario desde una fuente de datos externa.

Para renderizarlos se utiliza el software libre GNUPlot [4].

Manipulación de imágenes

Gimp (http://www.gimp.org.es/) se ha utilizado para las manipulaciones de imágenes que ha requerido este trabajo.

Gráficos UML

Una de las premisas en este trabajo es trabajar cuanto se pueda en ASCII.

Esta estrategia no sólo permite regenerar automáticamente todos los diagramas, y en consecuencia los documentos que los incluyan, si se decidiera, por ejemplo, cambiar el estilo, sino que permite una perfecta integración con GIT.

Esto incluye muchos de los diagramas UML utilizados, como diagramas de tiempo o de clases. El motivo es que puedan editarse fácilmente con cualquier editor, dedicando el esfuerzo a definir lo que tienen que representar y no cómo se va a representar.

El resultado final se delega en el "renderizador" utilizado, en este caso los programas gratuitos plantuml y websequencediagrams (http://plantuml.sourceforge.net/ y http://www.websequencediagrams.com/).

Para diagramas que requiren una presentación más cuidada y controlada, se ha utilizado Dia [5], que genera ficheros XML que pueden, también, ser procesados de forma automática.

Tests Unitarios

Los tets unitarios constituyen una valiosa herramienta para asegurar que el código está probado y que se cumplen las especificaciones.

Por ejemplo, en el desarrollo de un video juego es probable que una frame en HD tenga que renderizarse en menos de x milisegundos. Si no se utilizaran baterías de tests frecuentemente -continuamente, de hecho-, un desarrollador podría incorporar una modificación que no rompe la compilación, pero que dispara el tiempo de renderizado.

Con los tests units, este problema se conocería en cuanto se provoca y no quedaría oculto.

Python dispone de un excelente sistema de tests units, soportado por pytest (http://pytest.org/latest/). De hecho, veremos que es uno de los componentes fundamentales que incluiremos en el sistema de integración continua.

PEP8

Uno de las, a juicio de muchos, entre los que se incluye el autor de este trabajo, virtudes de python es que todo lo que está escrito en python tiene un aspecto "pythonico".

La idea es evitar los gustos particulares de cada programador y que todo el código, lo haya escrito quien lo haya escrito, tenga el mismo aspecto, lo cual facilita el trabajo en equipo.

PEP-8 (http://www.python.org/dev/peps/pep-0008/) define una "guía de estilo" de cómo deben escribirse los programas en python y la herramienta pylint (http://www.pylint.org/) ayuda a analizar el código reportando la calidad del mismo en base a PEP-8.

Intregración Contínua

"La integración continua (continuous integration en inglés) es un modelo informático propuesto inicialmente por Martin Fowler que consiste en hacer integraciones automáticas de un proyecto lo más a menudo posible para así poder detectar fallos cuanto antes. Entendemos por integración la compilación y ejecución de tests de todo un proyecto." [6]

Efectivamente, si es malo que se incorpore un error a la rama master, es mucho peor que dicho error quede oculto y que no se detecte lo antes posible.

Con sistemas de integración continua, evidentemente los errores sintácticos se conocen en el acto, pero también, si se han escrito los tests apropiados, los errores que de otra forma serían indetectables.

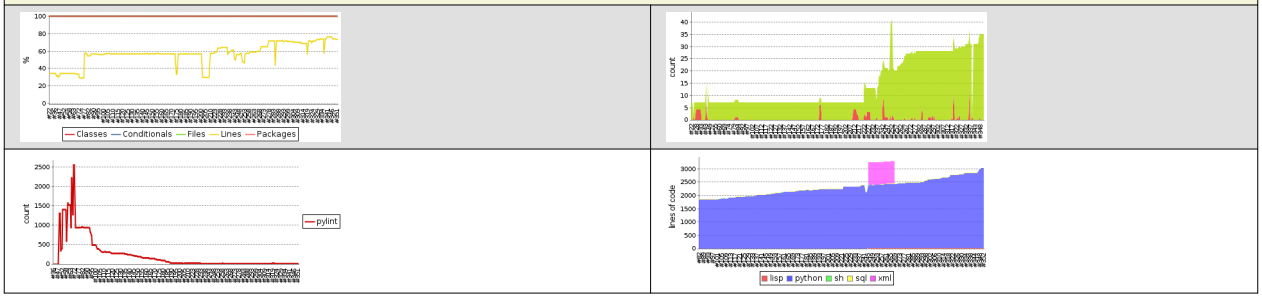
Además, permite tener una valiosa información en tiempo real sobre la calidad sintáctica del código escrito y sobre el nivel de cobertura del código por los tests escritos.

Jenkins [7] es una solución de software libre, realizado en java, que dispone de multitud de plugins que permiten un seguimiento muy preciso del desarrollo de software.

En este proyecto, tal como se muestra en la figura 31, cada vez que se hace un commit al repositorio de Git, este provoca una build en el servidor de Jenkins.

La build reporta no sólo si han pasado todos los tests, sino multitud de información sobre tiempos de ejecución, calidad del código en base a PEP-8 o cobertura.

Algunos diagramas de Jenkins



Metodología

Es imprescindible adoptar una metodología de trabajo. Desde mediados de los 90 [8] se comenzaron a definir métodos ágiles de desarrollo, como el famoso Scrum [9].

Todos ellos se basan en definir operativas ágiles y efectivas de desarrollo. No creo que ninguna sea mejor que otra. De hecho todas tienen sus puntos fuertes y puntos débiles. Lo que es seguro es que hay que adoptar alguna.

Y cumplirla.

No entraremos a discutir las cuestiones sobre ciclos, iteraciones, hitos, "dead lines" y demás cuestiones que quedan fuera del ámbito de este trabajo.

En lo que incideremos someramente es en el "ciclo típico de desarrollo", una vez que se ha definido un ticket que tiene que resolver una cuestión determinada.

Ciclo típico de desarrollo

Los desarrolladores tienen que desarrollar. Y tienen que hacerlo bien. En este trabajo se ha definido una metodología similar a las muchas existentes, para garantizar, dentro de lo posible que el trabajo de desarrollo es el correcto.

En primer lugar, y fuente de algún otro procedimiento en la metodología de desarrollo ágil utilizada, se genera un ticket que se registra en Trac.

Asignación

En algún momento, ese ticket se asigna a un desarrollador. Es posible que sea un responsable de proyecto el que asigne el ticket, si la metodología lo permite que sea otro compañero o que un desarrollador se lo autoasigne.

"Puesta al día"

El desarrollador debe asegurarse de estár en la rama apropiada del repositorio. Para ello hará, si es necesario un commit de su rama actual de trabajo y obtendrá del repositorio la rama sobre la que debe trabajar.

Resolución

El desarrollador trabajará en el ticket asignado. En muchas metodologías, un desarrollador no puede dejar un ticket hasta que lo haya cerrado. En otras, sin embargo, se permite conmutar de ticket.

Pruebas

Está prohibido que un desarrollador aporte al repositorio cambios que rompan la compilación.

El desarrollador debe crear los tests necesarios para probar en nuevo código y debe pasar todas las baterías de tests pertinentes para garantizar que sus cambios no introducen ningún error en su parte y otras partes del sistema.

Adicionalmente, deberá comprobar que su código cumple con el estilo de codificación mediante pylint.

Commit

Una vez que el ticket está resuelto, pasados los tests y comprobado pep8, el desarrollador podrá aportar los cambios al repositorio.

Jenkins

El repositorio de Git automáticamente provoca una build nueva en Jenkins, el servidor de integración continua.

En caso de que los tests no pasen, las personas implicadas serán automáticamente notificadas. En teoría, si se cumple el procedimiento, esto no debería pasar nunca.

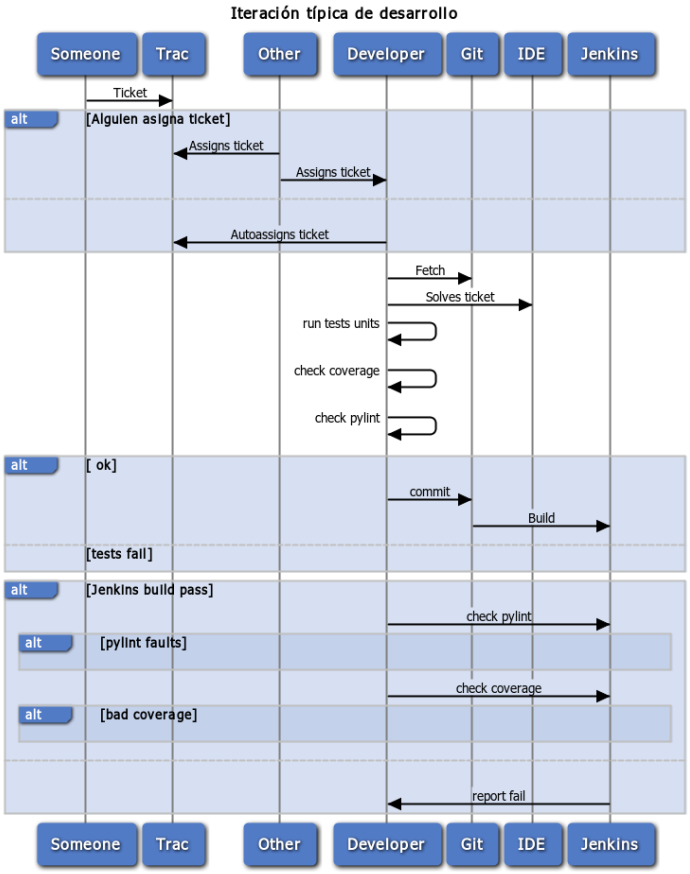


Fig. 28:: Diagrama temporal de interación típica

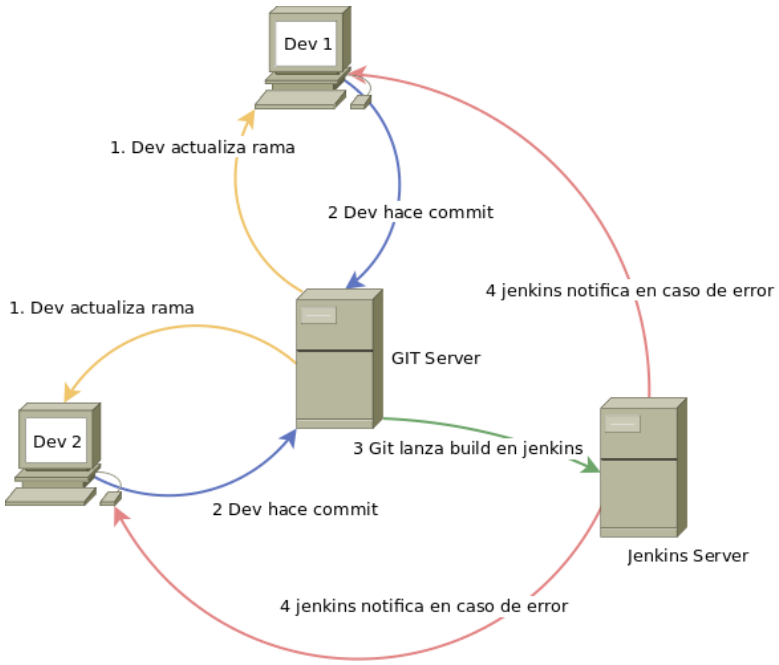


Fig. 29: Entorno de desarrollo

Diseño

Del análisis de los escenarios definidos durante la elicitación de requisitos, se desprende conveniencia de desplegar una serie de paquetes bien definidos.

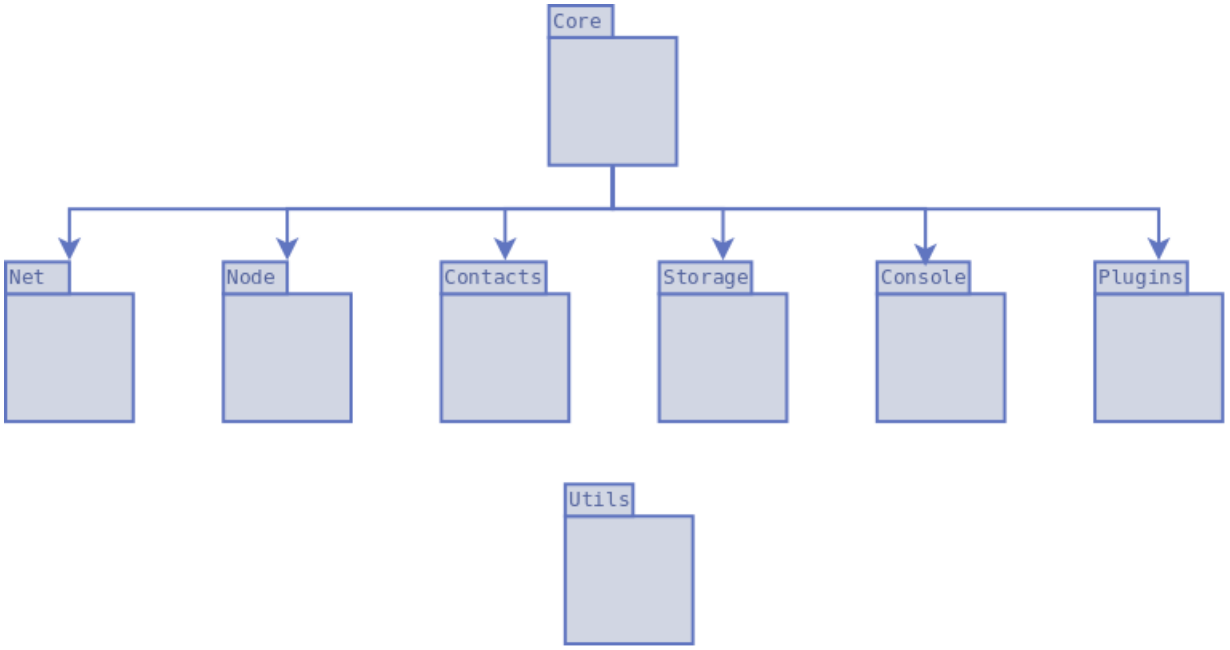


Fig. 30: Paquetes de ZOE

Core

Core es el núcleo de la aplicación, se encarga de levantar todas los componentes necesarios y de servir de conexión entre ellos.

Sus tareas más destacadas son:

- Instanciar cada uno de los módulos y plugins instalados y asegurarse de que siguen funcionando. En caso de caida de alguno de ellos lo instanciará de nuevo y se delega en el colector de basuras de python la destrucción del anterior.

Originalmente Core se dispuso como un Singleton, patrón que encaja perfectamente con su cometido, pero debido a la complejidad extra que introducía en las baterías de tests, se optó por descartar esa opción.

Utils

Paquete de utilidades genéricas que incluye clases y métodos de propósito general, utilizable desde cualquier punto de la aplicación tras hacer un import del módulo.

Console

Console proporciona uno de los dos interfaces de interacción con el sistema desde el exterior. En concreto, se deriva una clase de ella que es una consola por telnet, de manera que se puede hacer un telnet al puerto correspondiente y manejar el nodo, justo como se hace en mlDonkey.

Esta conexión TCP puede ser hecho por un humano, mediante un telnet, o por cualquier otro proceso automático.

Net

Soporta toda la funcionalidad de comunicaciones. Net instancia, al menos, un router, un transporte y un protocolo.

Net oculta completamente las cuestiones de red o comunicaciones al usuario o proceso que lo utilice.

Contacts

Paquete que gestiona la gestión de contactos, incluyendo envío o aceptación de invitaciones.

Storage

El paquete storage oculta la forma en que se almacenan o explotan los datos. Por defecto, ZOE utiliza un storage basado en Sqlite3, pero se puede intercambiar por cualquiera que proporcione con su interfaz.

Node

Paquete de alto nivel que gestiona envío y recepción de mensajes y solicitudes.

Plugins

Paquete diseñado para soportar plugins de terceros. ZOE se suministra con un plugin de ejemplo para que terceros puedan desplegar sus propios plugins en el sistema.

Detalle de Paquetes y diagramas de clases

Core

El paquete Core proporciona cuatro clases:

- Core:

  Al ejecutarse la aplicación se instancia y ejecuta una instancia de Core. Core es un un thread en cuyo "main loop" sólo comprueba si los elementos instanciados están funcionando correctamente.

- Publisher:

  Uno de los criterios de diseño en el sistema ha sido utilizar un patrón de "Publicador/Subscriptor". Esto facilita desacoplar los distintos elementos o futuros plugins instalados. A la vez que proporciona una manera muy sencilla de tener procesos asíncronos.

  La instancia de Core crea una instancia de Publisher que proporciona el único canal por defecto para publicar o subscribirse a publicaciones desde cualquier punto.

- TServer

  Otra importante decisión de diseño ha sido evitar los problemas típicos de la utilización de threads. En una implementación típica, suele ser habitual el lidiar con bloqueos y desbloqueos y esperar que no se produzcan "dead locks" ...

  En Zoe, los principales componentes son TServer, que no es más que un thread especializado que se comporta como un "despachador" de tareas, manejando tres colas con prioridades.

  Cualquier instancia que tenga acceso a un TServer le puede "encargar" que realize una tarea, sin esperar respuesta y enganchando a un callback a la finalización de ella, o esperando respuesta y con prioridad media, alta o baja.

  La correcta utilización de TServer elimina completamente el problema de los "racings" entre threads ya que cada TServer ejecuta su propio código con sus propios datos.

  Esto no sólo evita lidiar con los bloqueos. También hace que el código sea muy limpio y no se termine escribiendo código "spaguetti" donde cada thread intenta pelear por recursos que pertenecen a otros threads.

- ZObject

  ZObject proporciona a las clases que derivan de ella, unos métodos helper para utilizar el motor de publicaciones/subscripciones y asegura que las instancias tengan un nombre y acceso al Core.

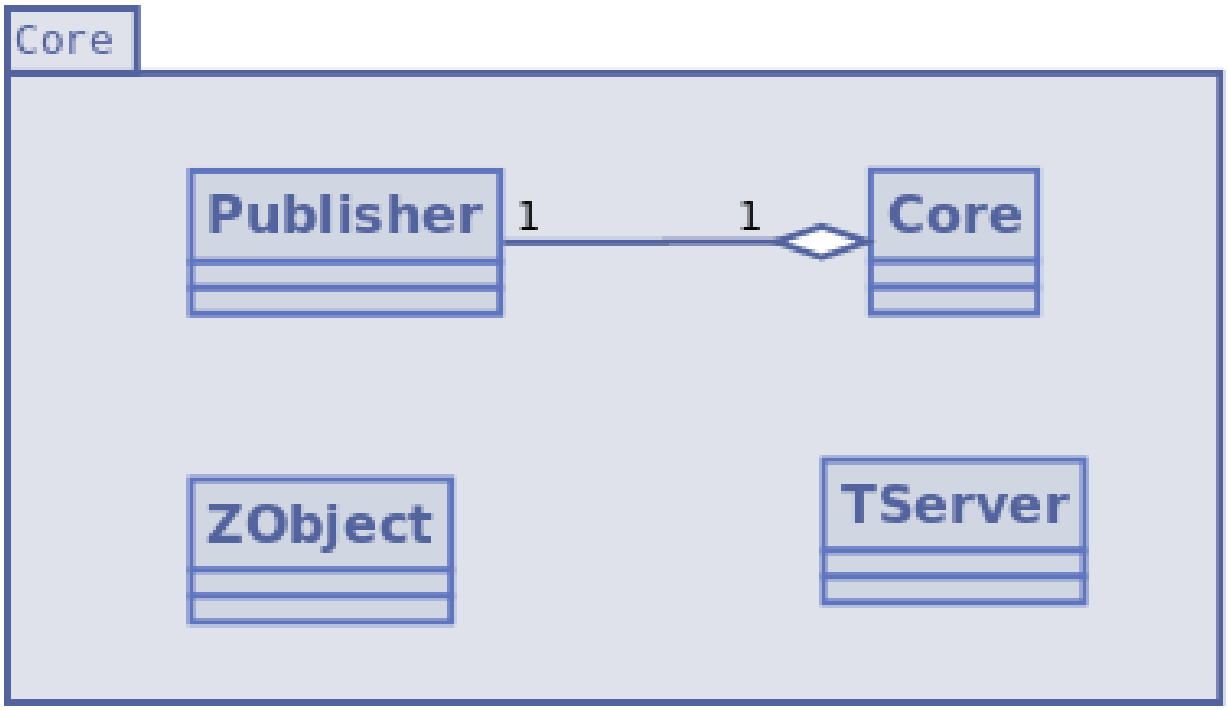


Fig. 31: Diagrama de clases del paquete Core

Utils

El paquete Utils suministra métodos de propósito general, como manejo de fechas, generación de UUIDS, cálculo de SHA1 y cinco clases que agrupan algunas funcionalidades específicas:

- Singleton

  Clase utilizada para definir instancias Singleton. Si bien el patrón de singleton es ampliamente discutido y tiene tantos detractores como admiradores, en ocasiones encaja perfectamente con los requisitos.

  De hecho, en este trabajo, originalmente, Core se diseñó como un singleton, puesto que sólo puede existir un core en la aplicación. Sin embargo, como ya se expuso, se eliminó puesto que complicaba enormemente la gestión de los tests units.

- Mailer

  Clase que proporciona un helper para enviar correos electrónicos a través de un MTA.

- Netutils

  Clase que aglomera utilidades de red como obtener lista de IPs de la máquina, determinar si una IP es de lan o de WAN, indicar si dos IPs están en el mismo segmento de red , etc ...

- ConfigManager

  Proporciona métodos para obtener cómodamente valores de ficheros de configuración ASCII.

- RSA

  Proporciona métodos para encriptar, firmar y desencriptar mensajes de un nodo a otro mediante clave pública/clave privada.

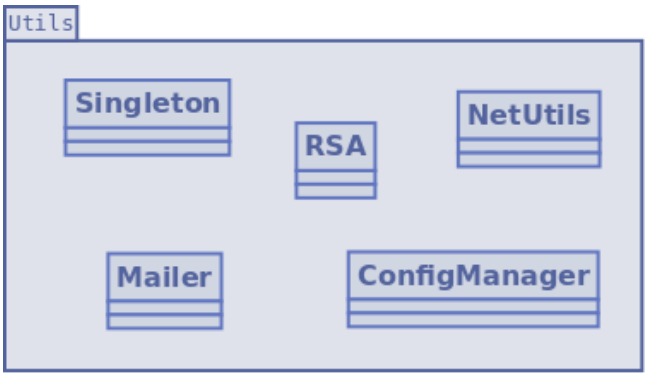


Fig. 32: Diagrama de clases del paquete Utils

Contacts

El paquete Contacts proporciona una sola clase que encapsula las acciones a realizar con contactos que no son envío o recepción de mensajes, como invitar, aceptar, buscar, etc

Deriva de TServer, que le proporciona la capacidad de compartarse de manera asíncrona como despachador de tareas y de ZObject que, como ya se indicó, le proporciona acceso al sistema publicador/subscriptor y al Core.

Al derivar de TServer, la máquina de estados de Contacts es la misma que la de TServer.

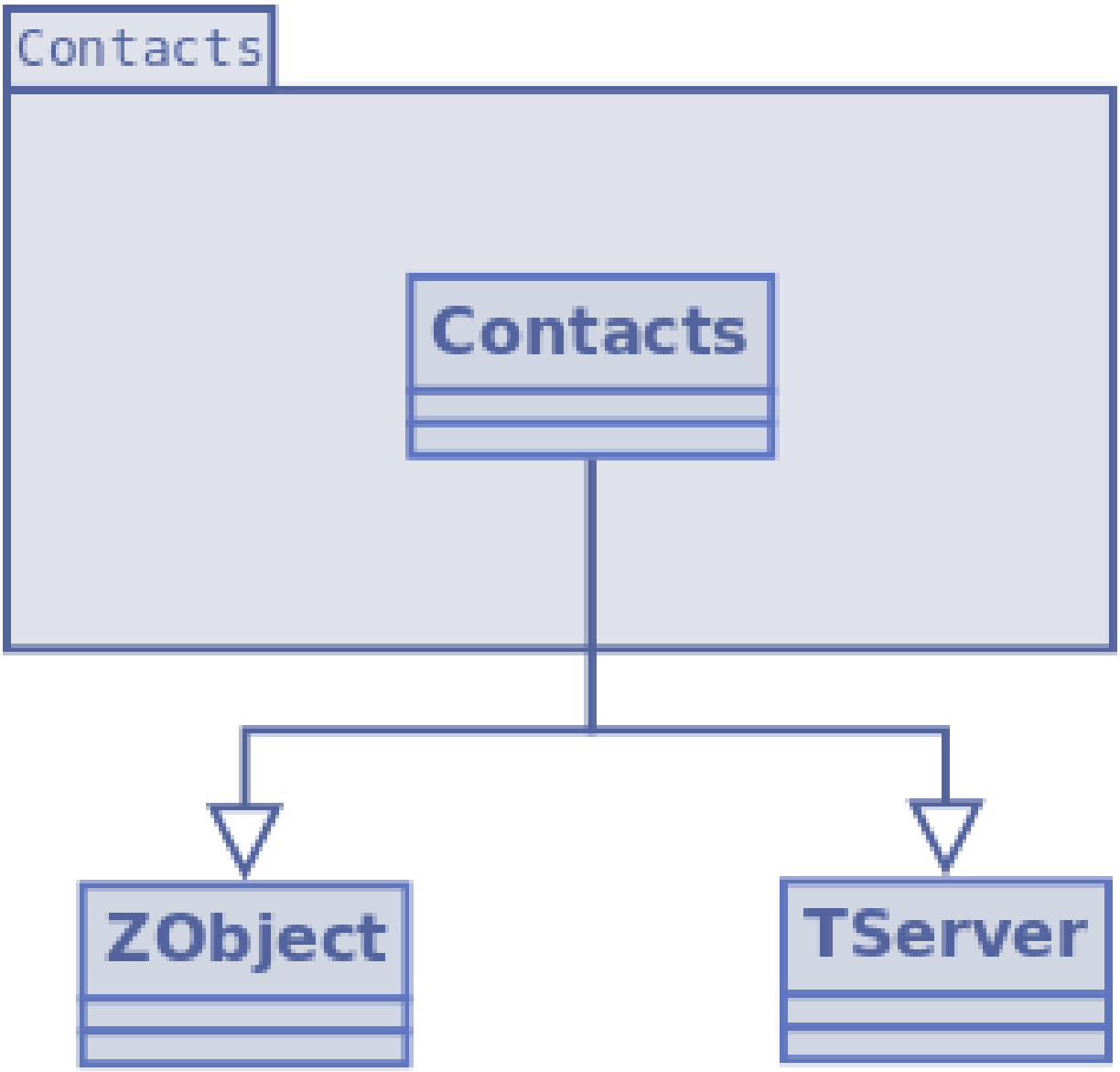


Fig. 33: Diagrama de clases del paquete Contacts

Console

El paquete console proporciona una de las principales características de ZOE ya que agrupa las clases que permiten desacoplar cualquier posible GUI de la lógica de la aplicación, en un patrón MVC y proporciona un método de operación sin GUI a través de un telnet.

La clase console proporciona los métodos básicos de cualquier consola. Incluye también un acceso, por defecto restringido, a un interprete embebido que da acceso a todos los objetos de la aplicación, diseñado, originalmente con efecto de debugging.

Adicionalmente instancia un proveedor de consolas telnet, de manera que como respuesta a un telnet, instancia una consola telnet específica por conexión. Dicha consola telnet puede ser extendida con métodos y atributos adicionales, como en el caso de ZoeConsole.

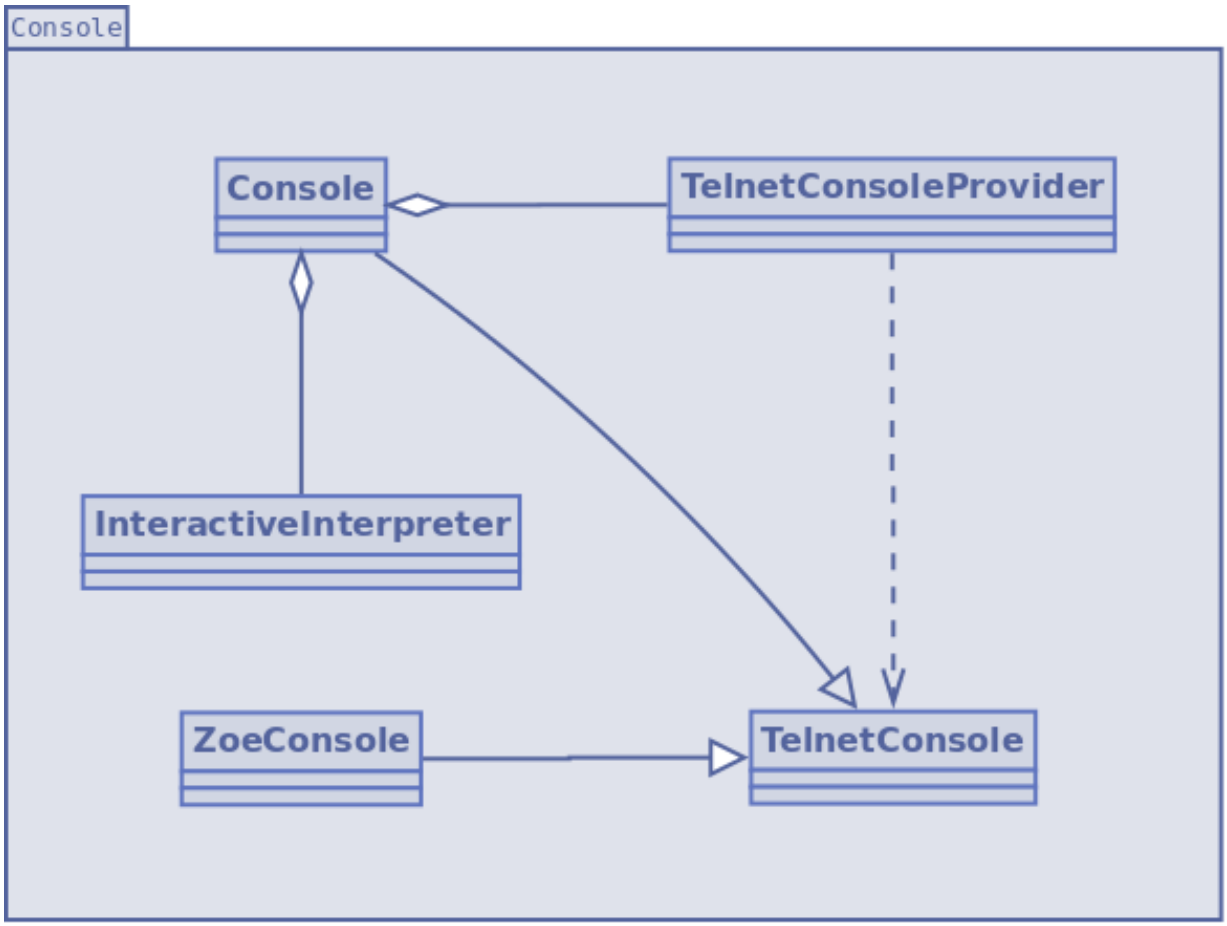


Fig. 34: Diagrama de clases del paquete Console

Node

El paquete Node proporciona una sola clase que proporciona métodos de alto nivel de actuación en la aplicación, como generación de nuevos mensajes, invitación de contactos o aceptación de invitaciones.

Adicionalmente, periódicamente interroga al storage por mensajes que estén pendientes de enviar, reintando su envío si procede.

Deriva de TServer, que le proporciona la capacidad de compartarse de manera asíncrona como despachador de tareas y de ZObject que, como ya se indicó, le proporciona acceso al sistema publicador/subscriptor y al Core.

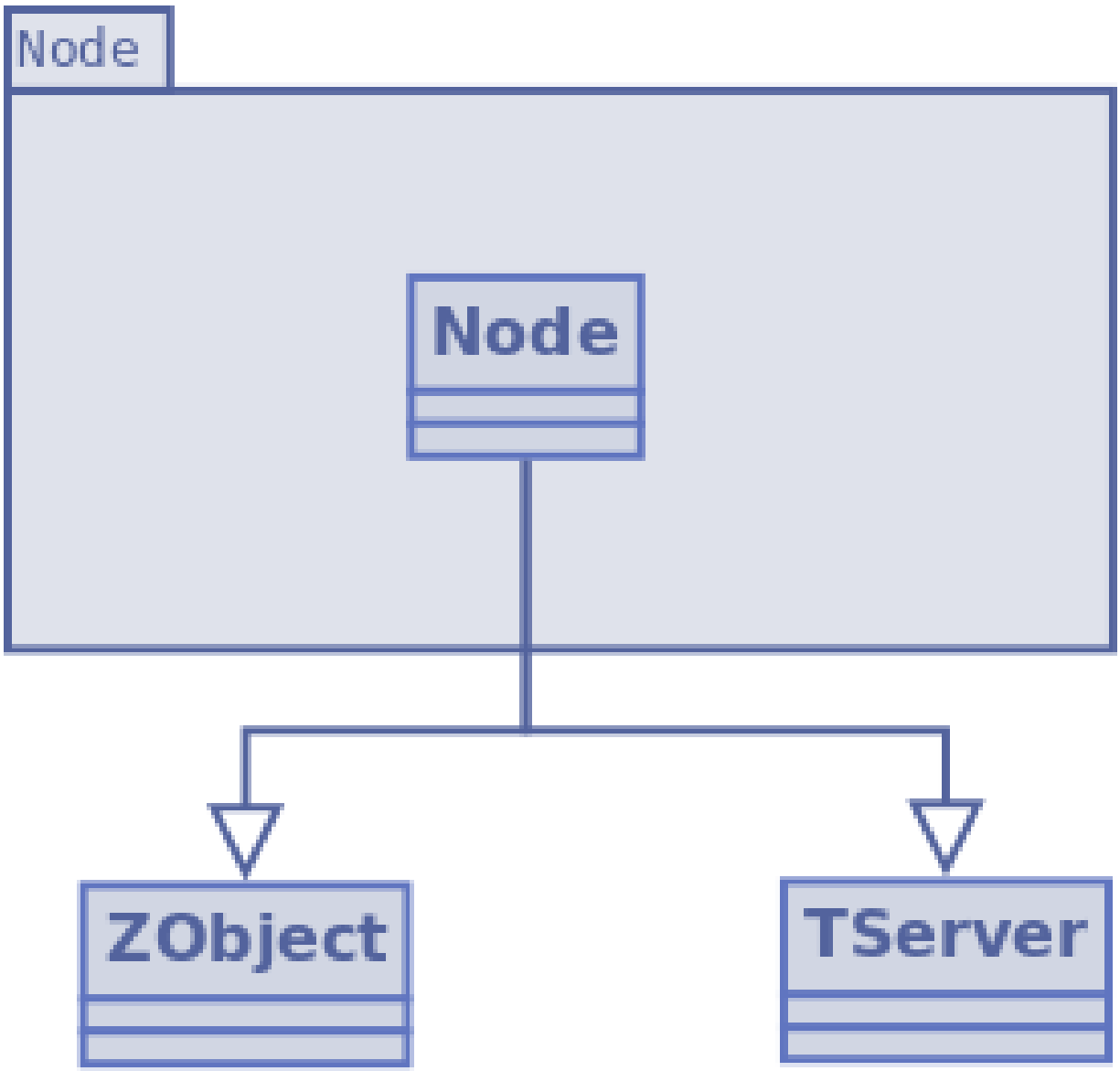


Fig. 35: Diagrama de clases del paquete Node

Net

Net proporciona la capa de comunicaciones de ZOE. Oculta completamente a otras capas y, por supuesto, a usuarios, cómo comunicarse con un nodo remoto.

Cuando la aplicación instancia el Core, este despliega una instancia de Net. Esta, a su vez despliega, al menos:

- Un Router. Encargado, básicamente, de mantener la lista de nodos conocidos con sus direcciones IP y puertos publicados y recibidos.
- Un transporte. Por defecto, y para poder realizarse UDP Hole Punching, Zoe utiliza un transporte basado en sockets UDP, pero podría ser reemplazado por cualquier otro.

  Zoe dispone un interface -Transport- que debe ser implementado. En este trabajo se implementará UDPTransport.
- Un protocolo. Por defecto, se utiliza el protocolo Zoe, que es simplemente una serialización del mensaje en cPickle.

  Zoe dispone un interface -Protocol- que debe ser implementado. En este trabajo se implementará ZoeProtocol.
- Un Firewall. No se implementará en el presente trabajo. Su interface por defecto se reduce a dos métodos: allow_in y allow_out.

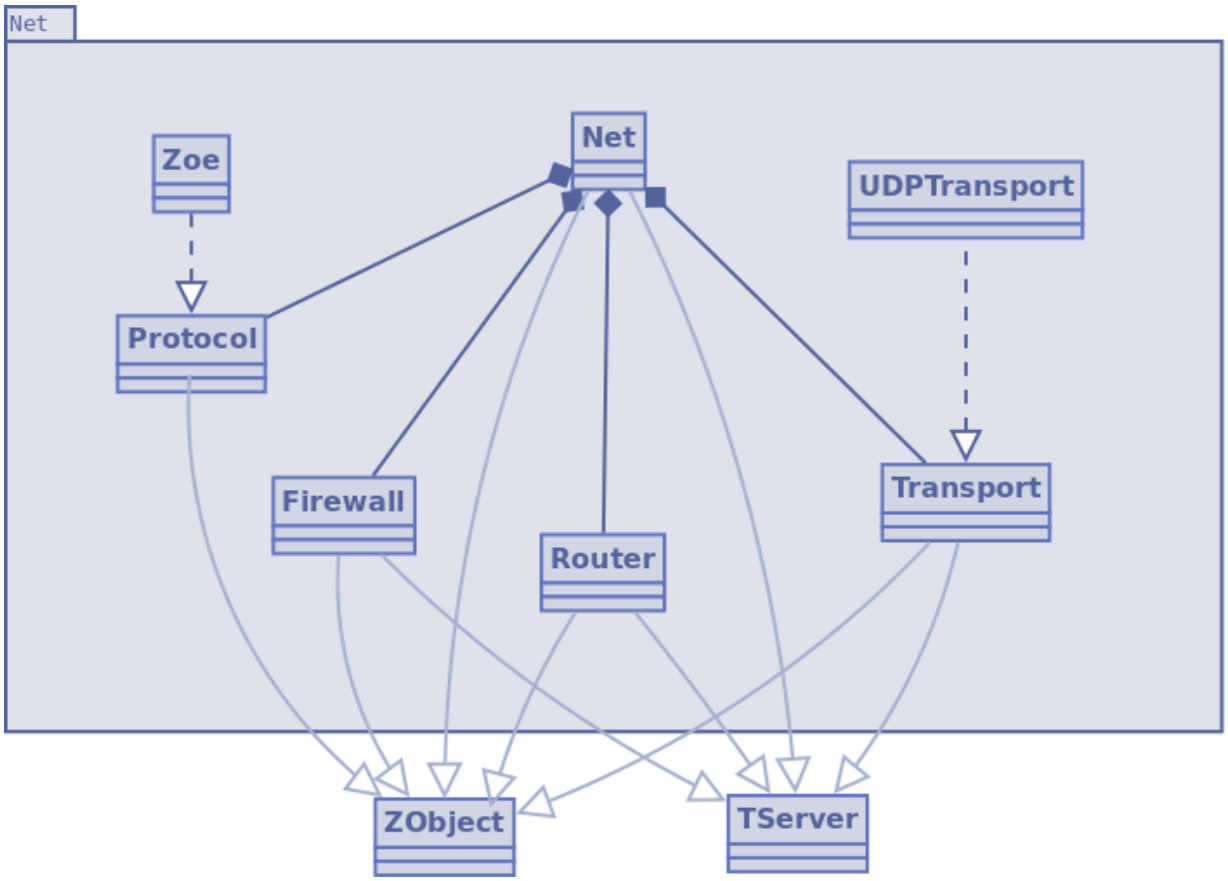


Fig. 36: Diagrama de clases del paquete Net

Storage

Storage encapsula todo el alacenamiento de datos persistentes en Zoe, como mensajes o información de contactos.

Se trata de un interface que ha de ser implementado para su utilización. En concreto, en Zoe se implementa una versión basada en sqlite, pero, siempre que se implemente el interfaz, el almacenamiento podría ser cualquier otro inventado o por inventar, como sistema de archivos, cualquier motor de bases de datos, etc ...

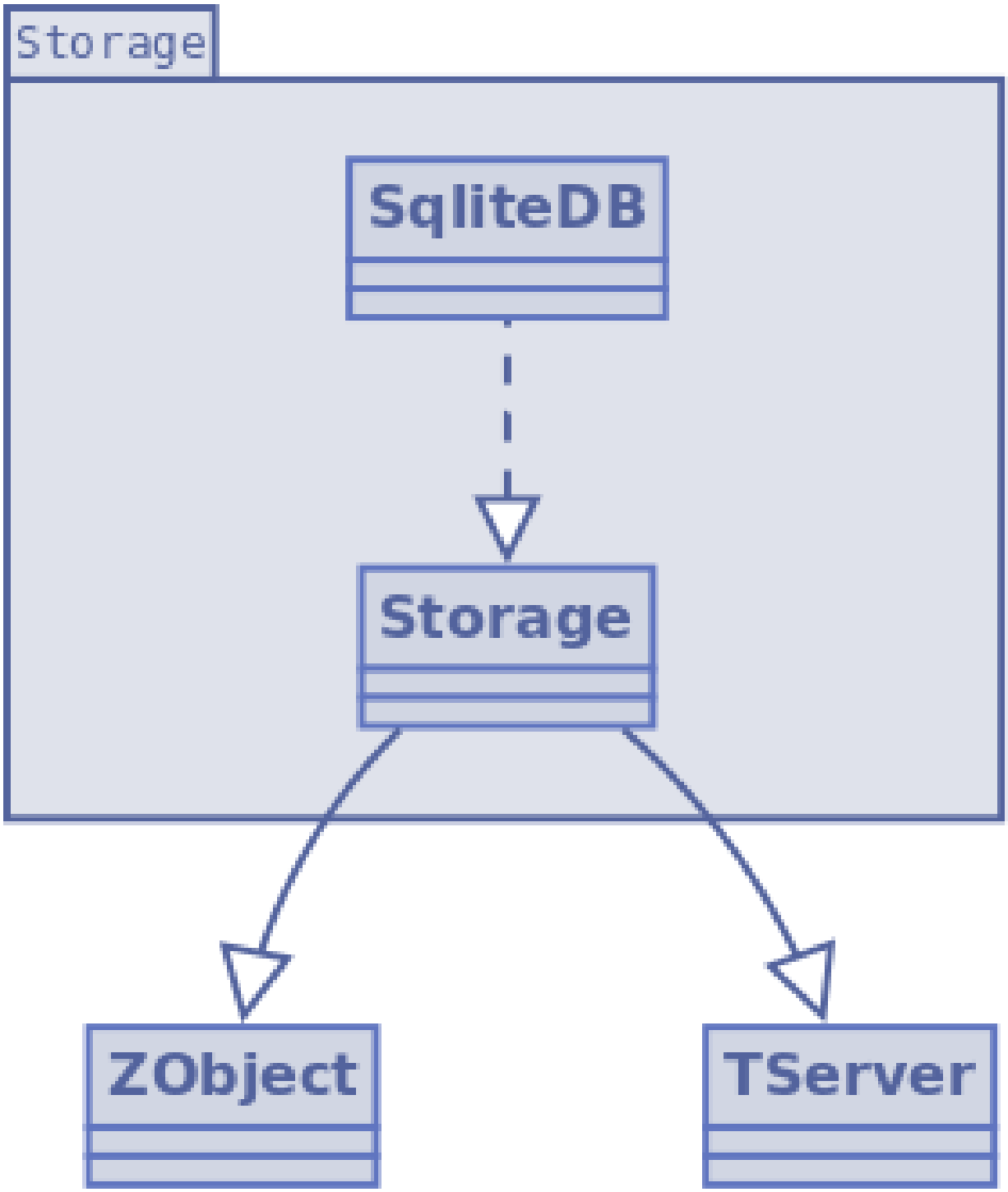


Fig. 37: Diagrama de clases del paquete Storage

Modelo

Cualquier storage que se derive de Storage debe dar soporte, al menos, a los datos persistentes que se requiren para el correcto fucionamiento de la aplicación.

Los requirimientos de persistencia de la aplicación son extremadamente austeras. En la implementación que se presenta, se almacenan contactos y mensajes.

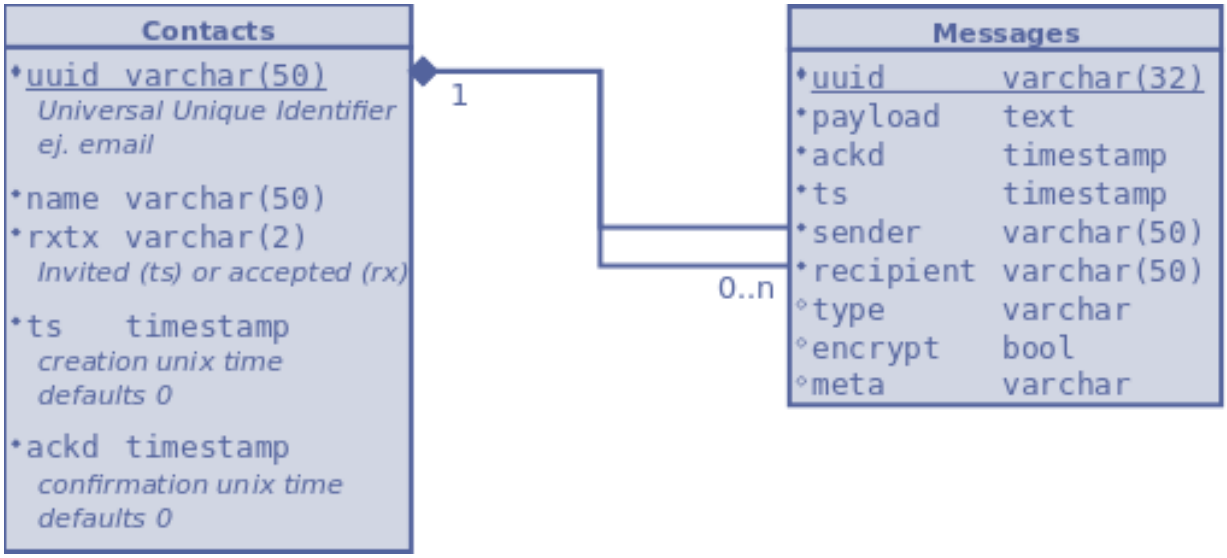


Fig. 38: Base de datos Zoe

Arquitectura Global del sistema

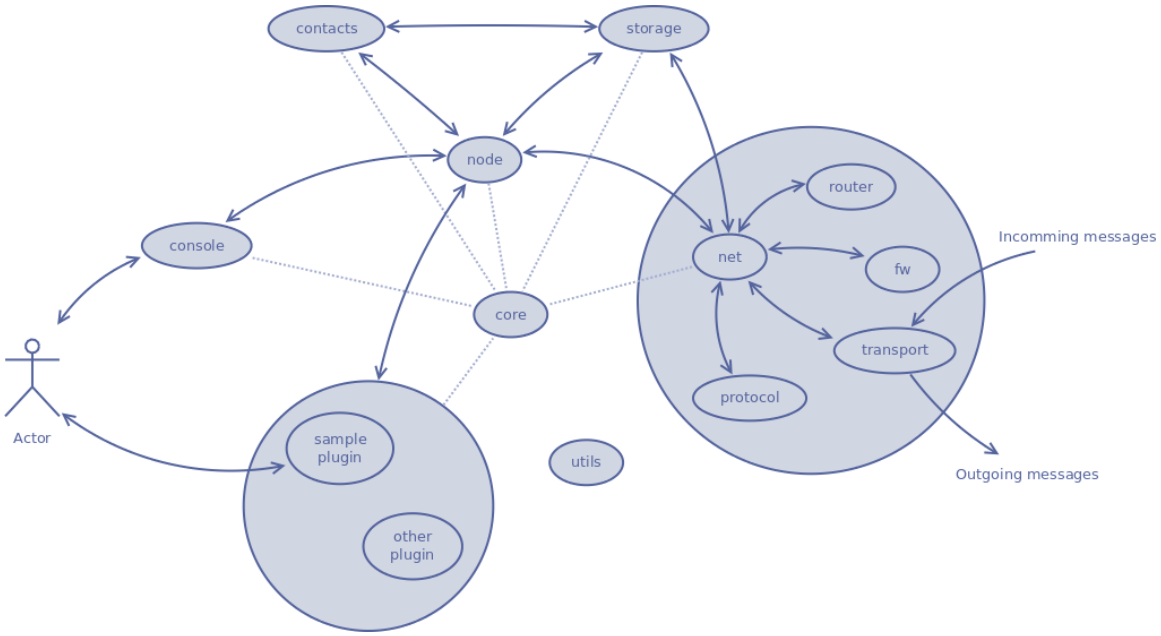


Fig. 39: Arquitectura global del sistema

Máquinas de estado

Cada uno de los componentes de la aplicación se comporta como una máquina de estados autónoma que reacciona a eventos provocados por ella misma o por otras.

A continuación se ilustra el funcionamiento como máquina de estados de cada uno de los módulos.

Core

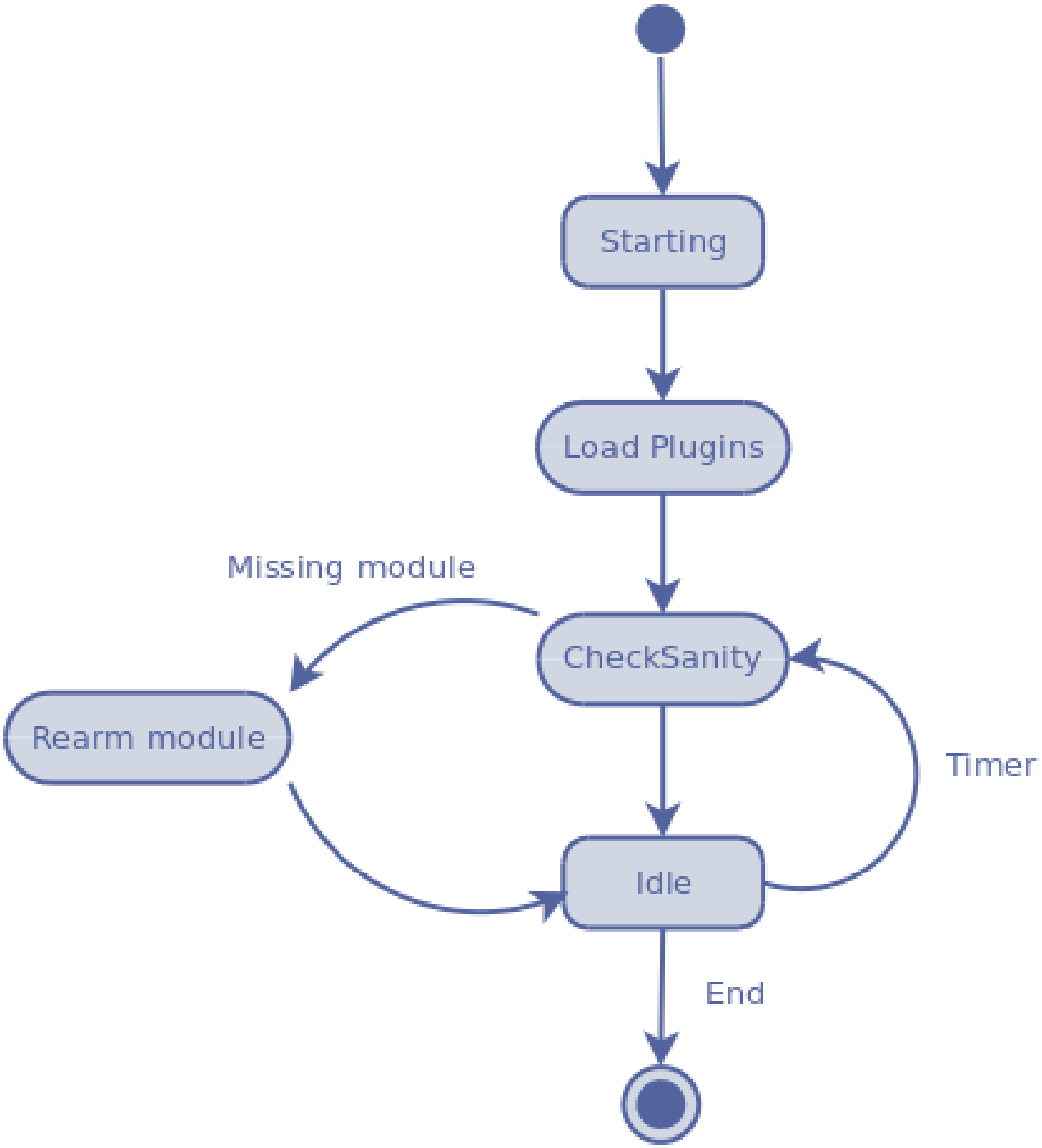


Fig. 40: Core SM

Console

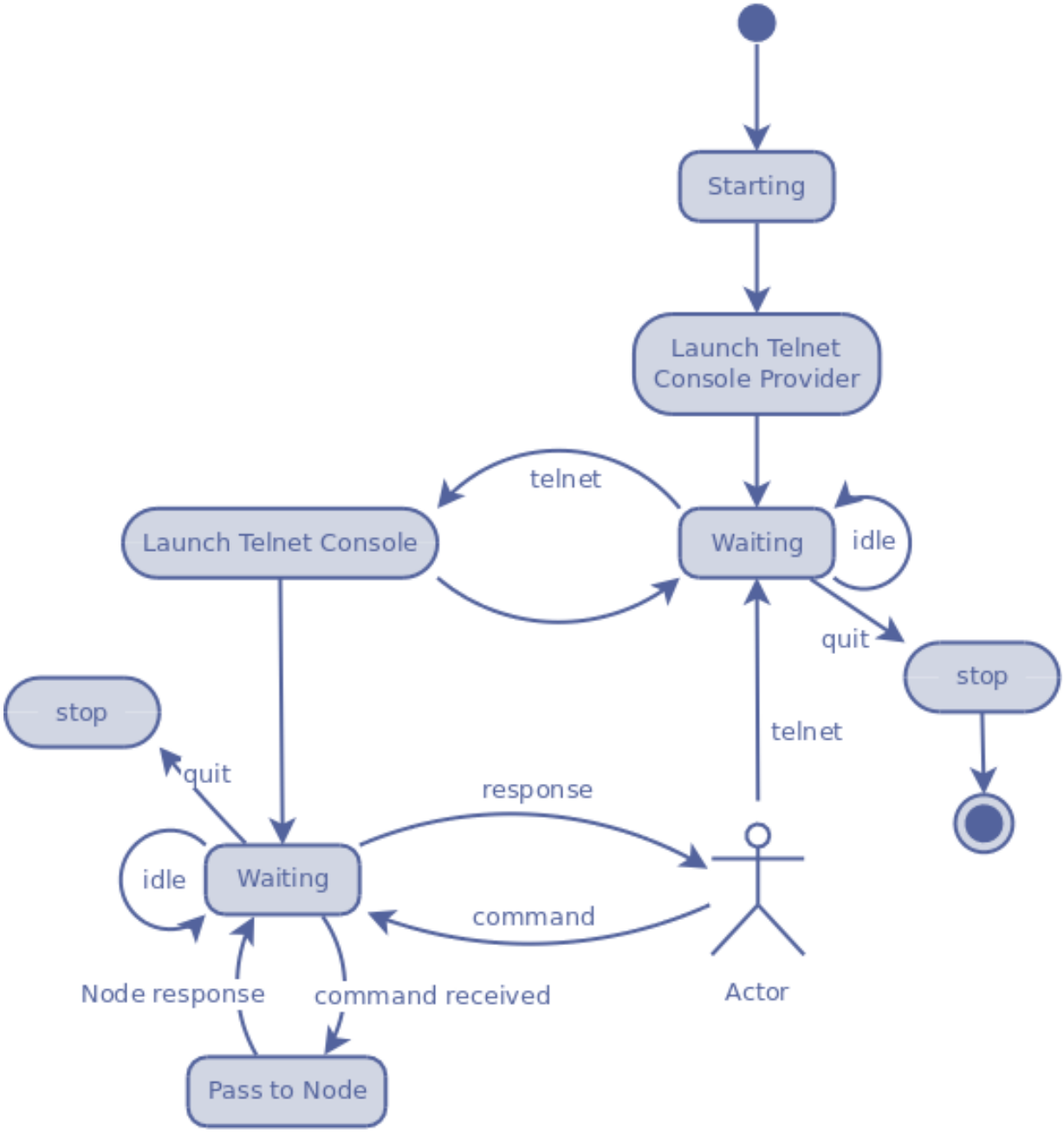


Fig. 41: Console SM

Contacts

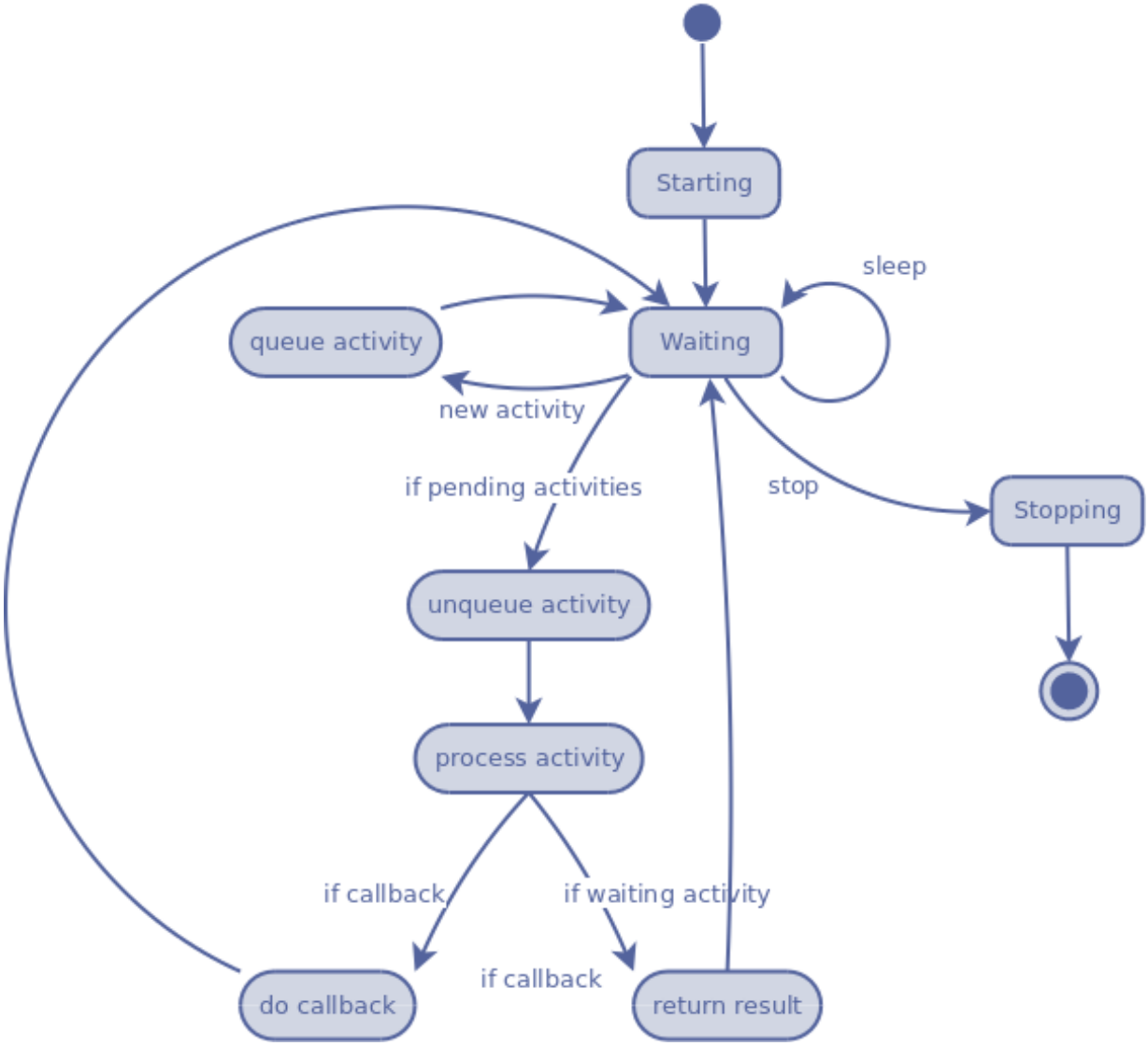


Fig. 42: Contacts SM

Node

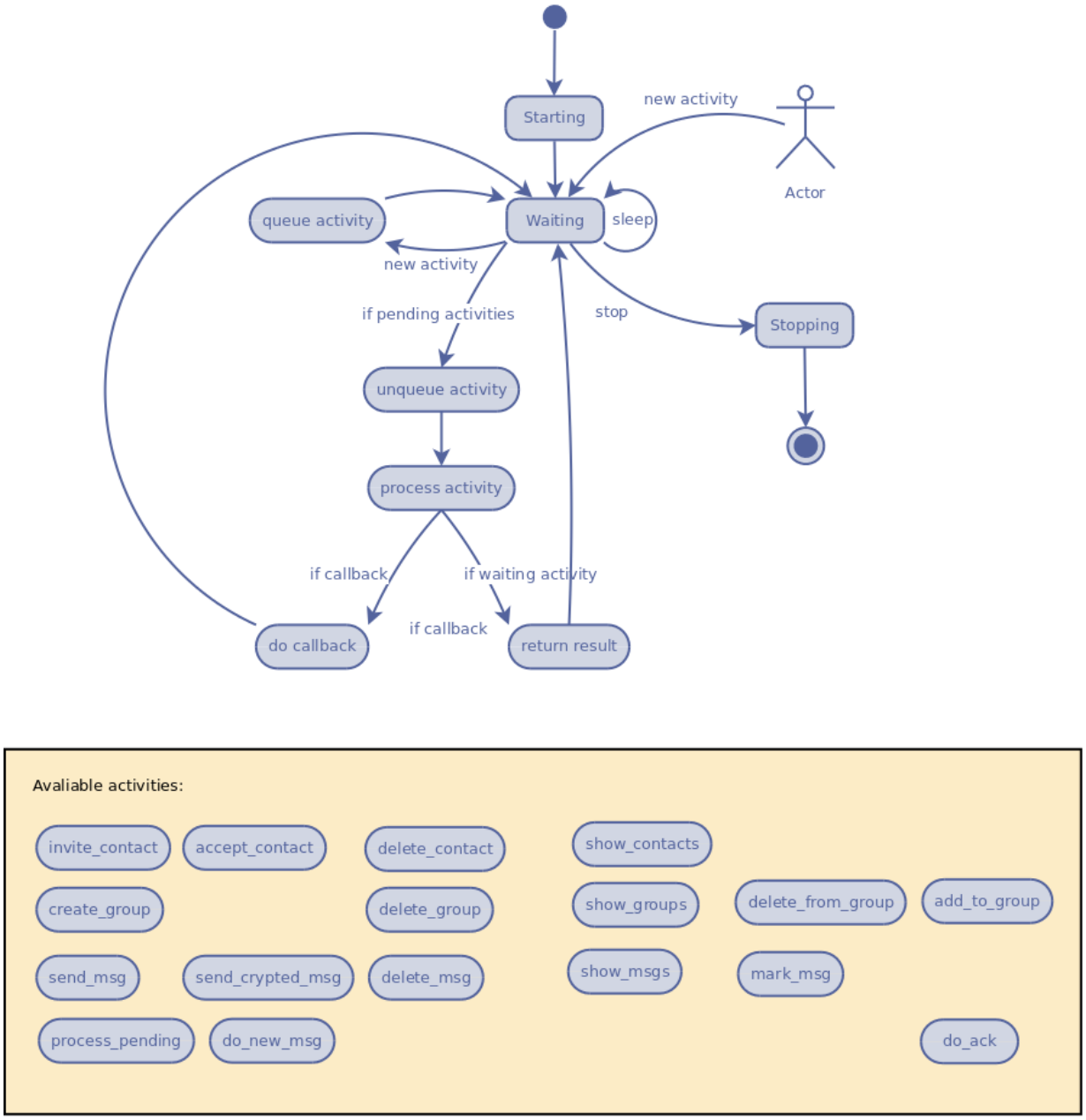


Fig. 43: Node SM

Patrones de diseño utilizados

"Una arquitectura orientada a objetos bien estructurada está llena de patrones. La calidad de un sistema orientado a objetos se mide por la atención que los diseñadores han prestado a las colaboraciones entre sus objetos."

"Los patrones conducen a arquitecturas más pequeñas, más simples y más comprensibles"

G
. Booch

MVC

"MVC consists of three kinds of objects. The Model is the application object,

> *the View is its screen presentation, and the Controller defines the way the user interface reacts to user input. Before MVC, user interface designs tended to lump these objects together. MVC decouples them to increase flexibility and reuse."*

[Design Patterns]: Elements of Reusable Object-Oriented Software (ISBN 0-201-63361-2)

Se trata de un patrón estructural.

Este trabajo desacopla la lógica del almacenamiento y de la representación gráfica. De hecho, en esta versión no se distribuye ningún GUI y se delega la construcción de estos a terceros.

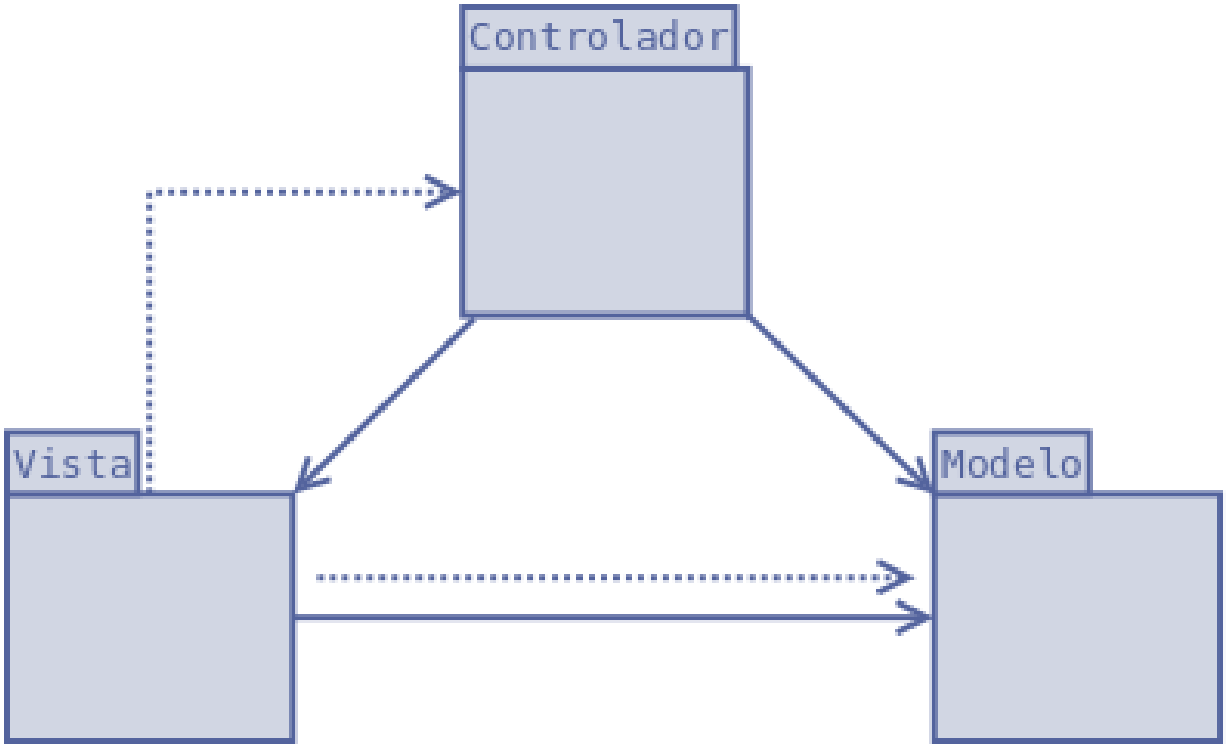


Fig. 44: Patrón MVC

Facade

Facada es un patrón estructural que provee un único interfaz unificado y simplificado a interfaces internos en el sistema. Define un interface de nivel superior que hace más fácil la utilización del sistema.

Node se ajusta al patrón de diseño Facade, puesto que su tarea es, principalmente, exponer una capa simplificada de las funcionalidades más internas del sistema.

"The facade pattern is ideal when working with a large number of interdependent classes, or with classes that require the use of multiple methods, particularly when they are complicated to use or difficult to understand. The facade class is a "wrapper" that contains a set of members that are easily understood and simple to use. These members access the subsystem on behalf of the facade user, hiding the implementation details. [10]"

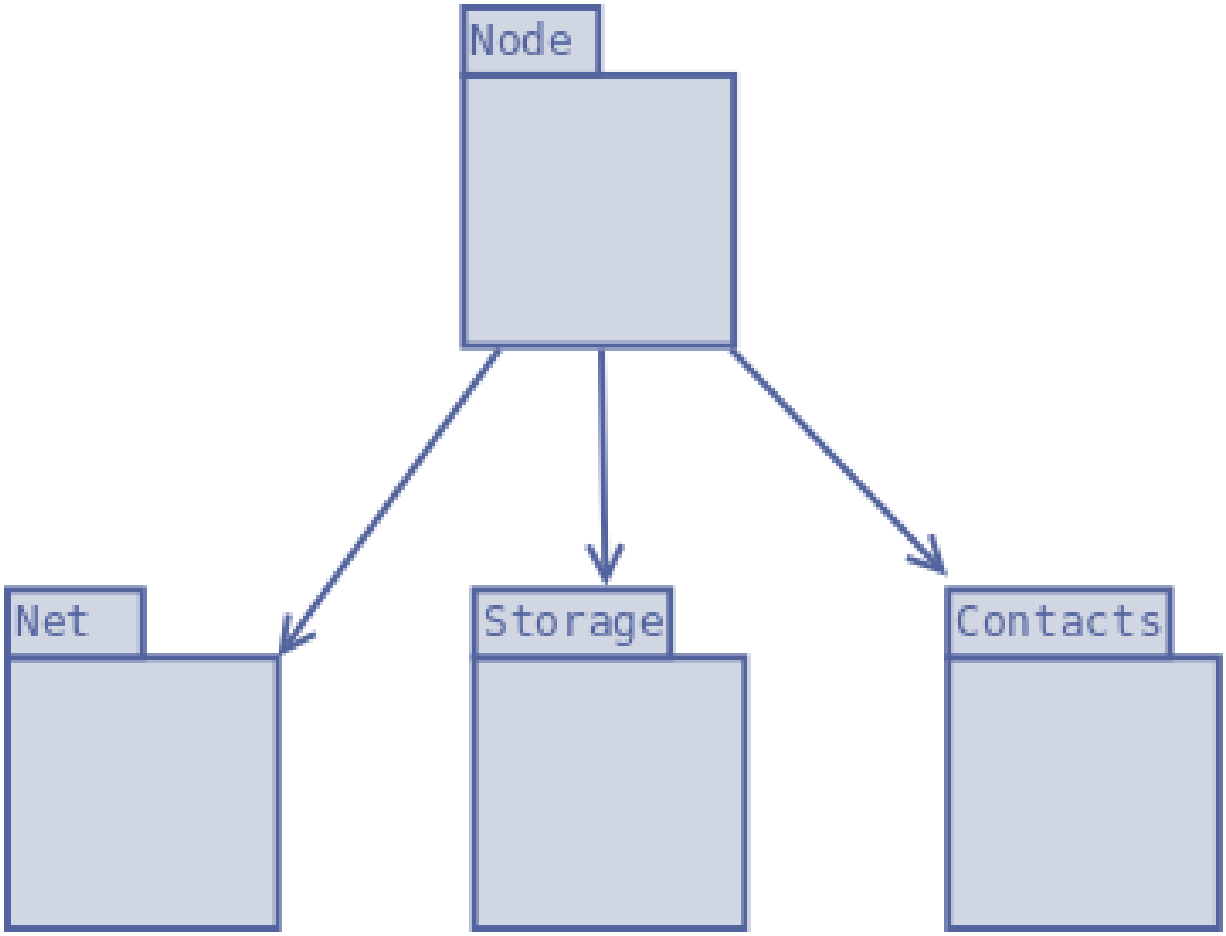


Fig. 45: Patrón Facade de Node

Servidor de actividades

- Nombre: Servidor de actividades
- Tipo: Comportamiento
- Intención: Evitar problemas inherentes a los threads. Proporcionar asincronía.
- Aplicación: Colaboración en sistemas multithread.

La utilización de este patrón evita que el código de un módulo tenga que ejecutar código de otro módulo con su propio "hilo de vida".

En este patrón, en lugar de que, desde un módulo, se ejecuten métodos públicos de otro módulo, se encola en el segundo una actividad síncrona o asíncrona. El módulo que tiene la actividad encolada, la ejecuta en su propio hilo, evitando problemas de "racings" entre hilos.

Adicionalmente, proporciona asincronía ya que el servidor de activades puede, opcionalmente, invocar un callback del módulo solicitante.

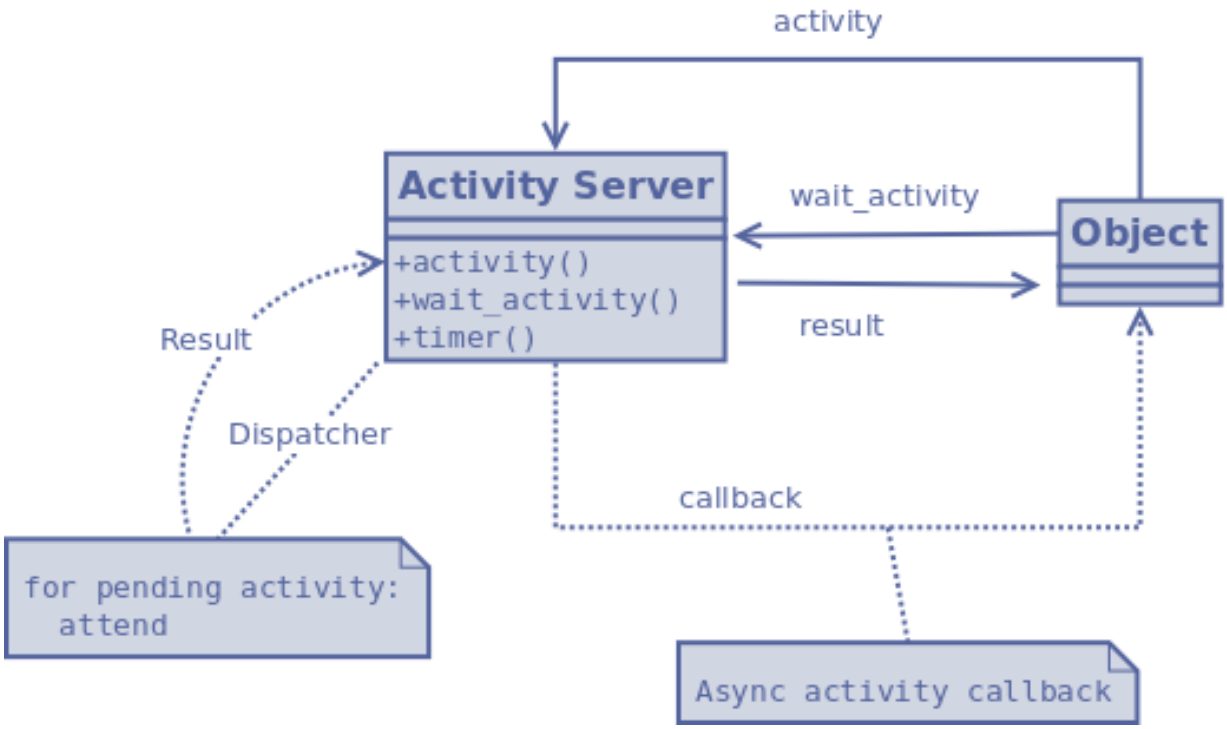


Fig. 46: Patrón Servidor de Actividades

Código de ejemplo usando el patrón Servidor de Actividades:

```
class A(tserver.TServer):

  ...

  def time_wasting_method(self, **kw):
      time.sleep(84600)

  def quick_callback(self, **kw)
      self.activity(method=self.time_wasting_method, **kw)

  def some_method(self, **kw):
      who = kw.get('who')
      who.activity(method=who.do_b_things, callback=self.quick_callback)

class B(tserver.TServer):

  ...

  def do_b_things(self, **kw):
      ...
      # do hard work
      ...

 if __name__ == "__main__":

    a = A()
    b = B()

    a.some_method(who=b)
    a.do_otherthing()
```

Publicador/Subscriptor

También llamado Observer [11], el patrón Publicador/Subscriptor facilita que las partes del sistema estén desacopladas y que las cosas simplemente "pasen".

El core de la aplicación despliega un Publicador al que se realizan subscripciones. Estas subscripciones incluyen:

- Quién se subscribe.
- Regular Expresion a la que se subscribe.
- Callback a llamar.

Ejemplo

```
self.subscribe('net/new_message/*', self._do_new_message)
```

En este caso, se trata de una subscripción para que cuando cualquiera publique un mensaje que cumpla la regular 'net/new_message/* se llame al callback _do_new_message.

Como ya se ha indicado, el publicador asertará si el callback tarda demasiado, por lo que normalmente, ese callback debe crear una actividad nueva en el objeto llamado.

Para evitar mal uso del subscriptor, incluye un control de tiempo de manera que el callback llamado no puede durar más de unos pocos milisegundos.

Para completarlo, en cualquier lugar del sistema, se pueden pegar "gritos" que puedan ser de interés para alguien.

Cada vez que alguien publica algo, el publicador comprueba que subscriptores tienen subscripciones con expresiones regulares que se ajusten a la publicación y llama a sus callbacks.

```
self.publish('net/new_message/darth.vather@gmail.com', **msg)
```

Cualquiera que se haya subscrito a net/new_message/* o net/new_message/dart*, o con cualquiera otra expresión regular que haga "match", será invovado al callback de su subscripción con todo el contenido del mensaje. En nuestro caso, el método _do_new_message del subscriptor sería llamado con el contido del mensaje.

Esta es una manera extremadamente sencilla y eficiente de que partes del sistema "se enteren" de cosas que pasan.

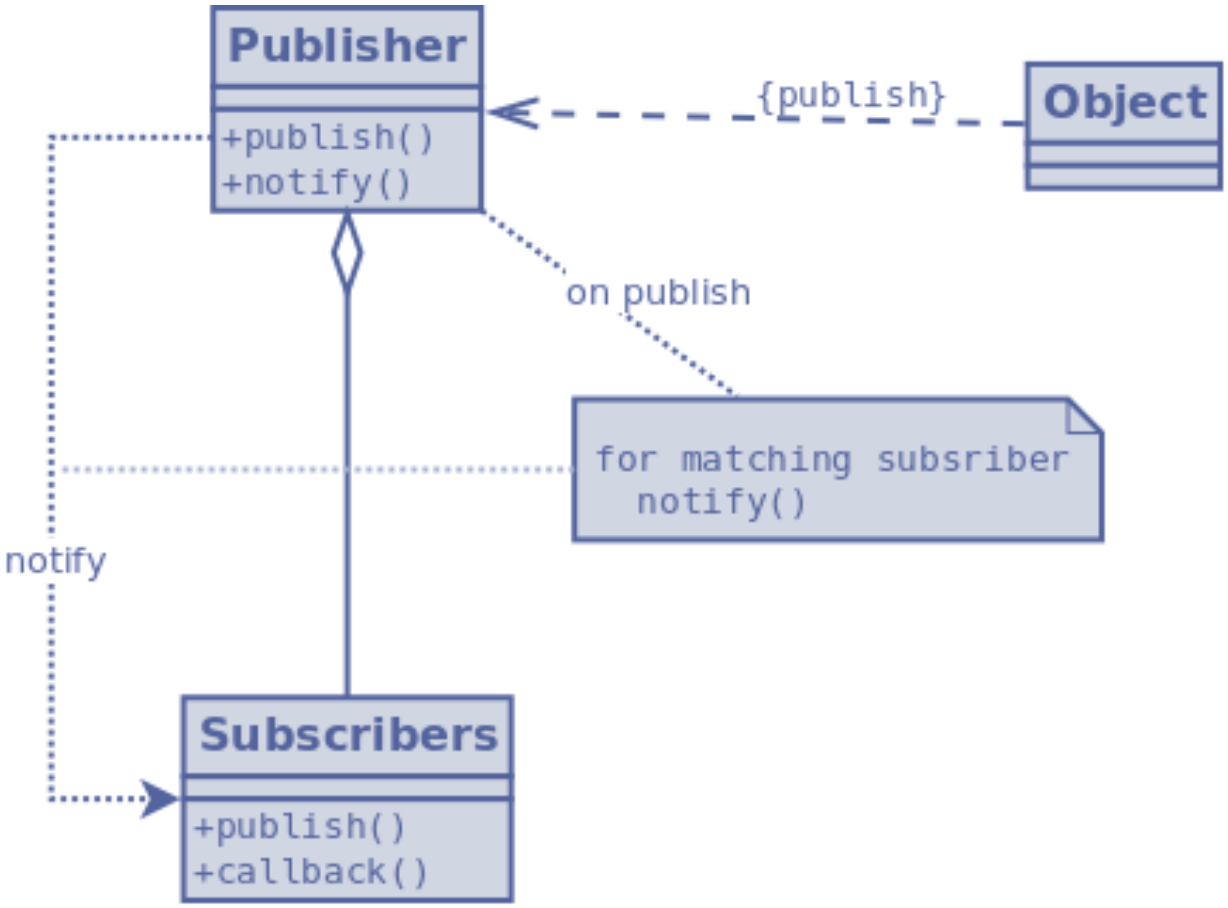


Fig. 47: Patrón Publicador/Subscriptor

Referencias:

- http://siul02.si.ehu.es/~alfredo/iso/06Patrones.pdf
- http://hillside.net/patterns/
- Design Patterns: Elements of Reusable Object-Oriented Software (ISBN 0-201-63361-2)

Algoritmos funcionales principales

Invitar Contacto

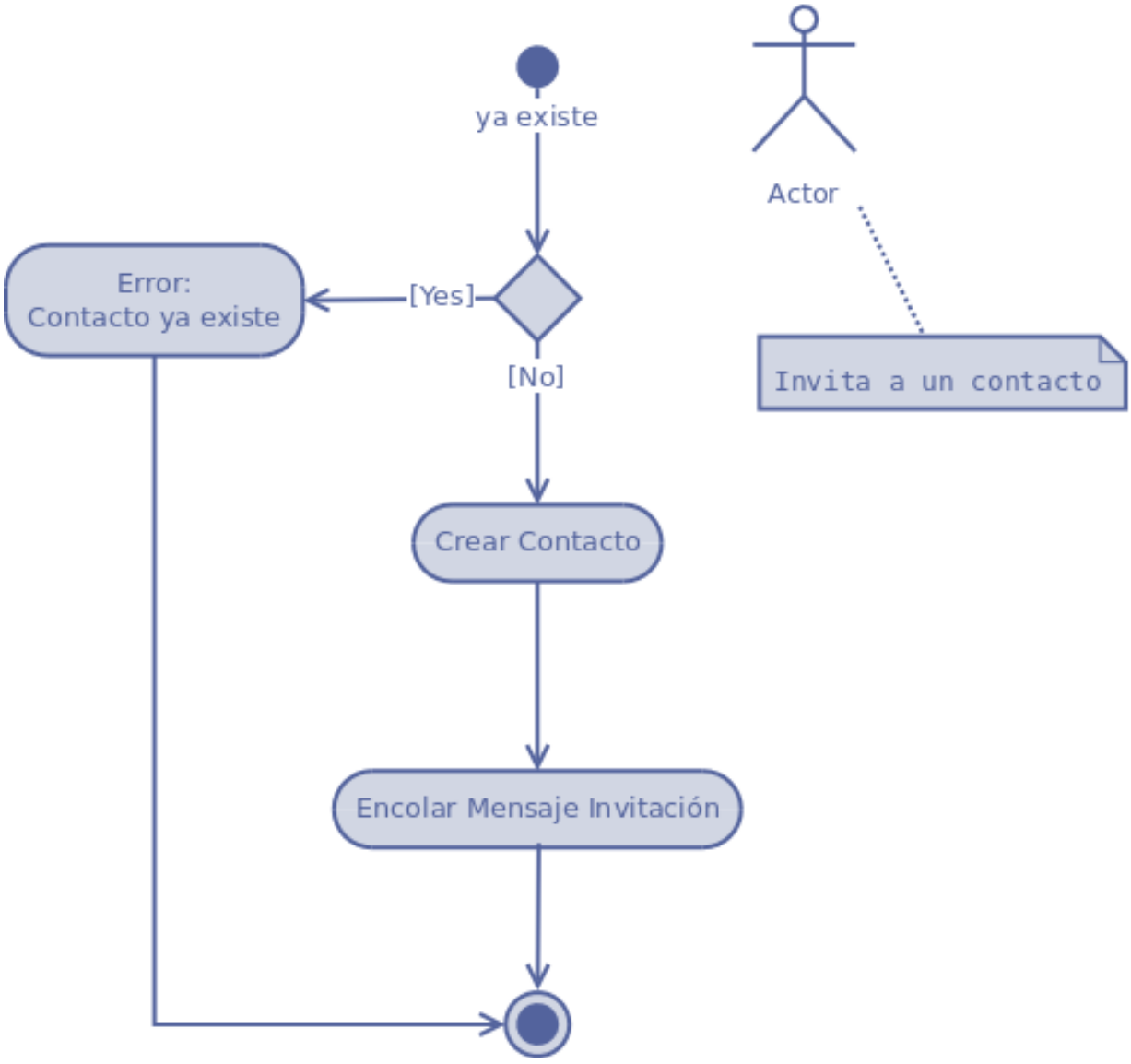


Fig. 48: Diagrama de Actividades Invitar Contacto.

Aceptar Contacto

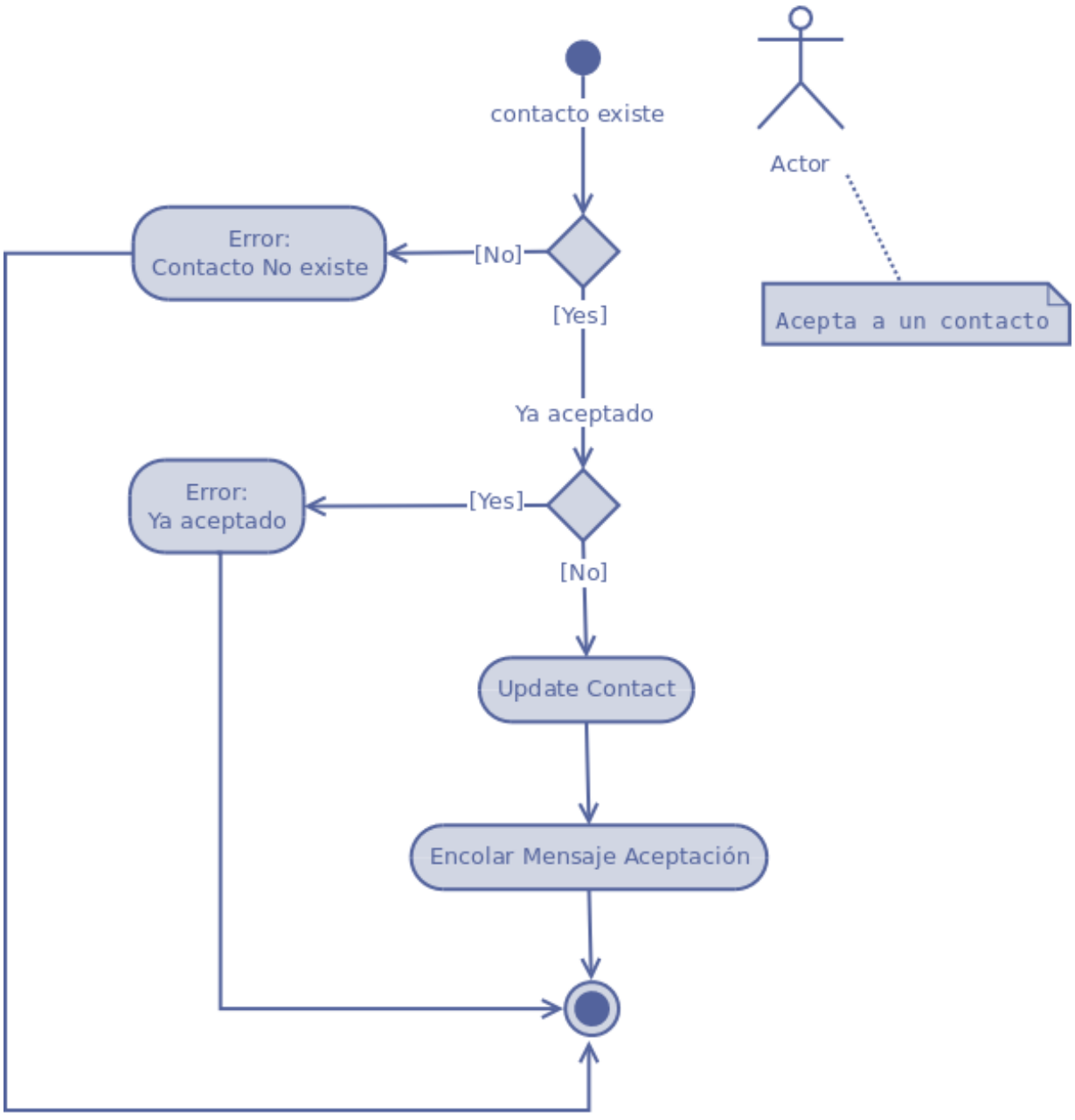


Fig. 49: Diagrama de Actividades Aceptar Contacto.

Mensajes

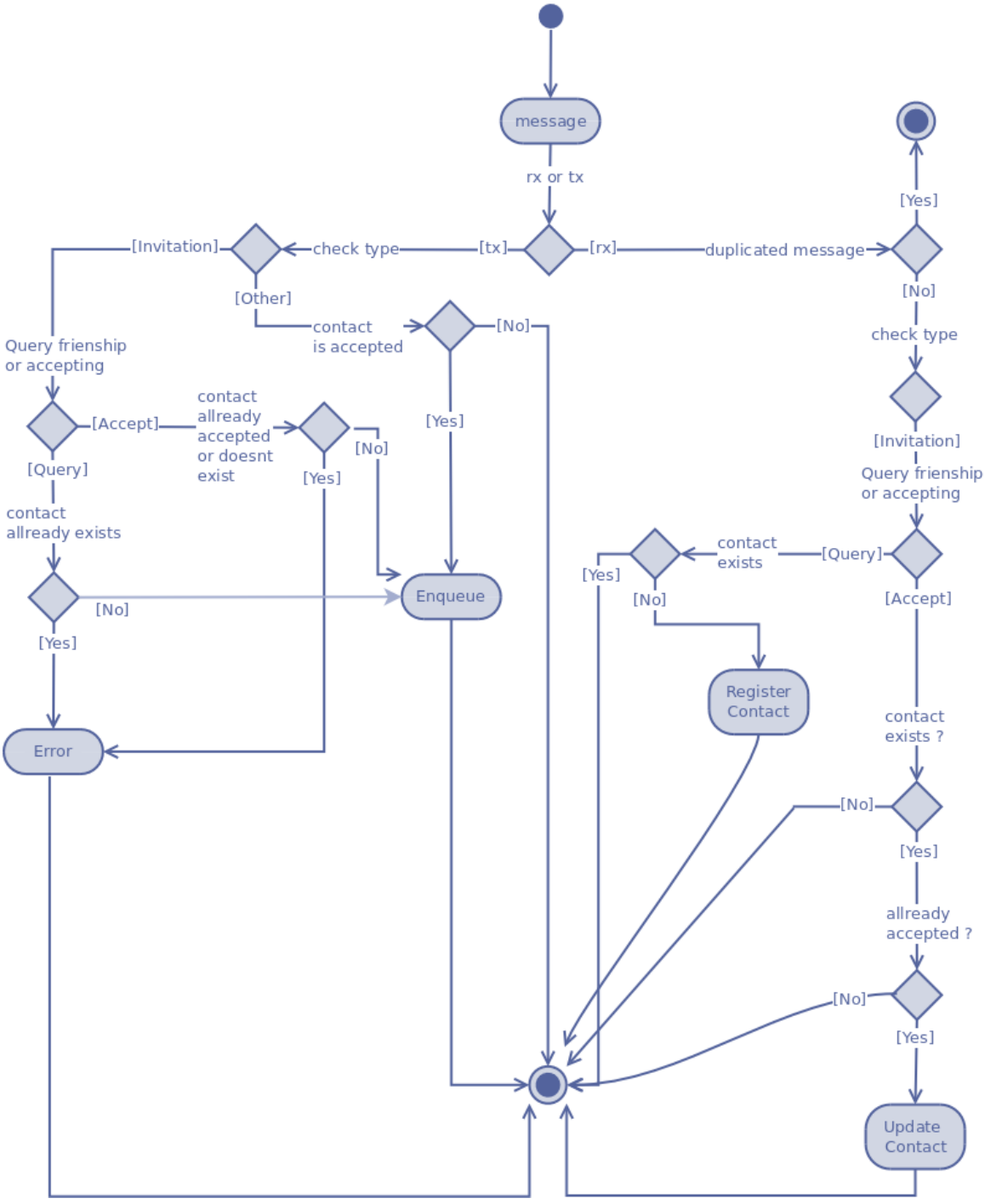


Fig. 50: Diagrama de Actividades Mensajes

Algoritmos no funcionales principales

TServer

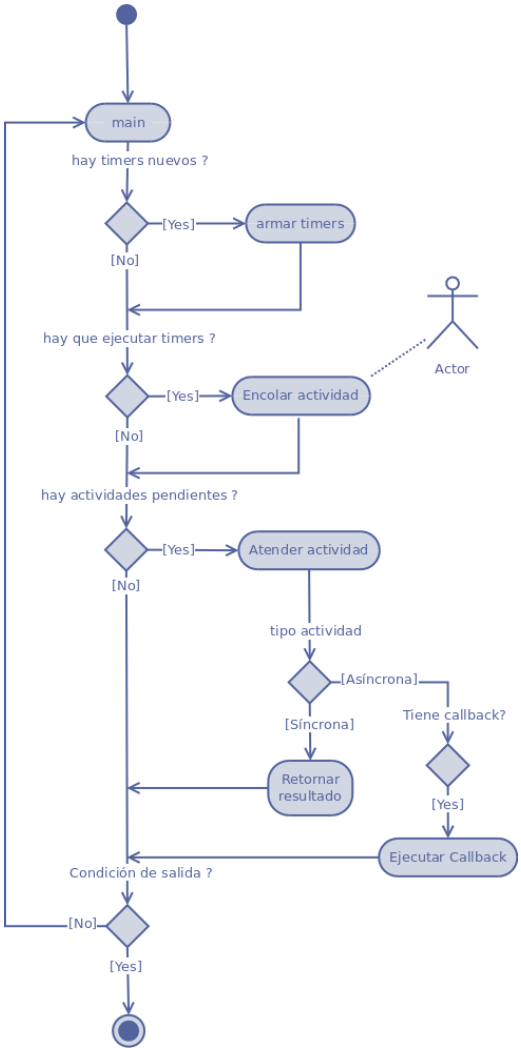


Fig. 51: Diagrama de Actividades de todos los TServers

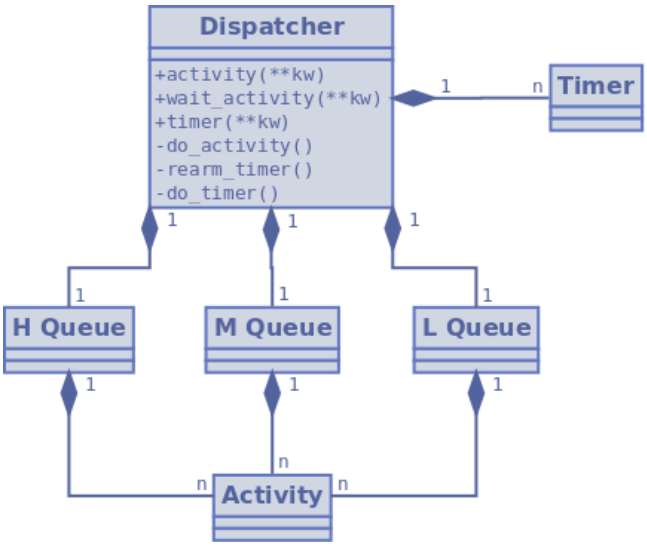


Fig. 52: Diagrama de clase de TServer

Publisher

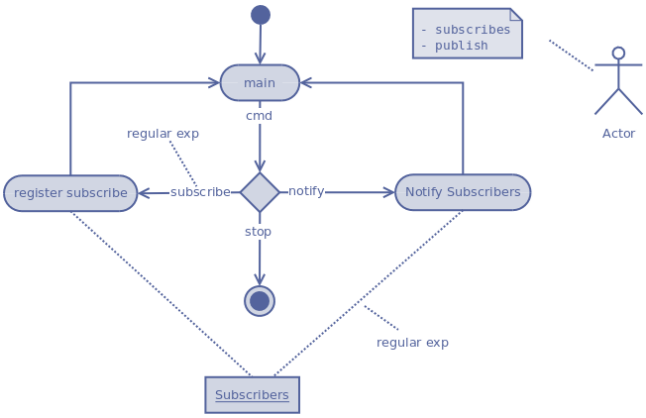


Fig. 53: Diagrama Actividades de Publisher

Encriptación RSA

En un sistema de comunicaciones privado las comunicacione deben ser privadas.

Esto no sólo significa que los mensajes deben ser P2P. Estos tienen que ir encriptados de forma que si un tercero intercepta los mensajes mediante una técnica "Man in the middle" [12] , o suplantando identidad de uno de los extremos, no pueda entender los mensajes.

En ZOE se utiliza un ancriptación RSA [13] de 512 bits

Funcionamiento de claves asimétricas RSA:

- Cada usuario tiene un par de claves:
  - una privada que sólo el conoce
  - una pública, que deben conocer los usuarios con los que quiera comunicarse.
- Si A encripta con su clave privada, sólo aquellos que tengan la clave pública podrán leerlo y tener garantía de que A generó el mensaje. Sin embargo, A no tiene garantía de que su clave pública, proporcionada a sus contactos, no llegue a terceros.
- Si A encripta con la clave pública de B, sólo B podrá interpretar el mensaje, pero puesto que B no tiene control sobre cual es la propagación de su clave pública, no puede tener garantía de que A fue quien generó el mensaje.

Solución:

- A encripta el mensaje con la clave pública de B. Sólo B podrá desencriptar el mensaje.
- A firma el mensaje con su clave privada. Cuando B desencripte el mensaje, podrá utilizar la clave pública de A para comprobar la firma.

Si C encriptara el mensaje con la clave pública de B haciéndose pasar por A, cuando B intente comprobar la firma con la clave pública de A, fallará, quedando garantizado el origen del mensaje en caso de éxito.

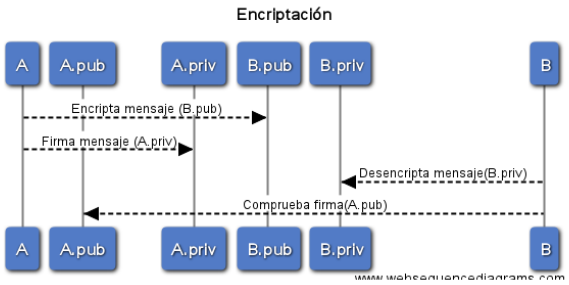


Fig. 54: Encriptación RSA

Discover/Punch

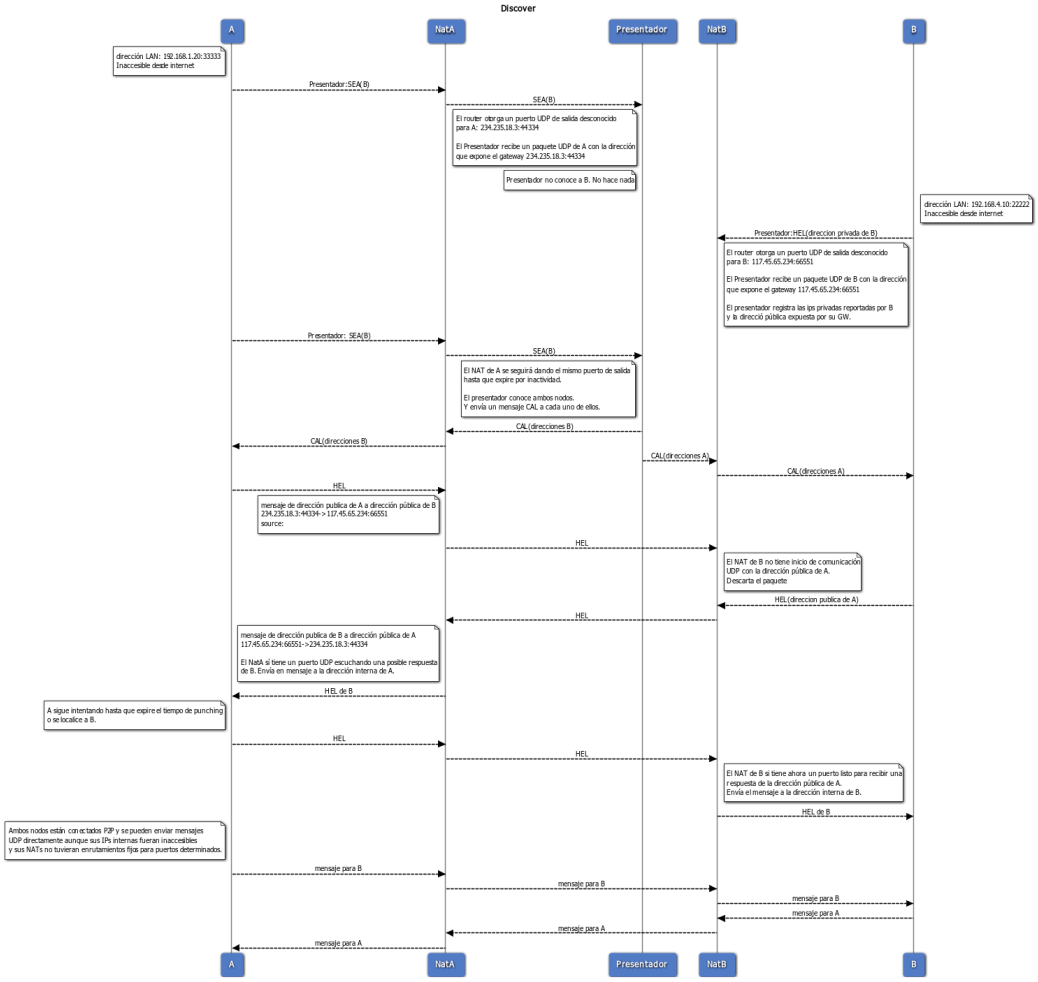


Fig. 55: Punching

Fundamentos técnicos

El problema de las comunicaciones P2P es que, al menos, uno de los extremos a comunicar debe ser accesible en IP y puerto.

La solución obvia, es que uno de los extremos tenga una IP pública y los puertos necesarios accesibles. Dicha IP pública, en principio, debería ser fija o, al menos, resolvible por DNS usando cualquier proveedor de dyndns, de manera si cambia la IP pública, inmediatamente se propaga el cambio en los DNS del mundo.

Los usuarios, en sus casas y dispositivos móviles tienen una IP pública, bien fija bien dinámica.

En los hogares, podrían configurar reglas de NAT en su ruter, que suelen permitirlo, para que los paquetes alcanzaran el dispositivo interno.

En los dispositivos móviles con conexión directa de datos, los puertos estarán abiertos en función del operativo y operador.

En ambientes empresariales, o en entornos wifi, los equipos de los usuarios son, en principio, inaccesibles. No es fácil que un administrador enrute el tráfico de nuestra aplicación favorita a nuestra ip: puerto internos ....

Y esto se debe al uso de enmascaramiento para acceder a internet.

Es enmascaramiento es una técnica utilizada para poder tener muchos equipos internos accediendo a internet (o en cualquier caso, a otra red), con una sóla IP pública. Esto se hace así ya que las IPs públicas son escaras y escasas ... en IPv4 [#_ipv4]_sólo hay 2^32 IPs públicas, muchas de las cuales corresponden a redes privadas y aunque sean más de 4000 millones, grandes corporaciones y paises acaparan grandes segmentos del espacio de direcciones.

Con enmascaramiento, los dispositivos gateway tienen unas tablas donde identifican ips internas y puertos internos. Cuando llega un paquete de internet y otra red, el dispositivo enmascarador deshace el enmascaramiento y reenvía en contenido a la ip interna: puerto interno.

Quizás algún día, cuando IPV6[#ipv6]_ se estandarice, cada equipo electrónico del mundo podrá tener una IP pública ...

IPv6 admite 340.282.366.920.938.463.463.374.607.431.768.211.456 (2^128 o 340 sextillones de direcciones), lo que supone que en una población mundial de 7.000 millones de personas, cada una de ellas podría tener 4.8×10^28 direcciones públicas !!

En tanto cada persona tengamos 4.8*10^28 is públicas para nuestros dispositivos, gracias al comportamiento de los NATs, es posible implementar la técnica UDP Hole Punching para establecer conexiones P2P entre equipos con IPs privadas y puertos desconocidos.

NAT

"NAT (Network Address Translation - Traducción de Dirección de Red) es un mecanismo utilizado por routers IP para intercambiar paquetes entre dos redes que asignan mutuamente direcciones incompatibles. Consiste en convertir, en tiempo real, las direcciones utilizadas en los paquetes transportados. También es necesario editar los paquetes para permitir la operación de protocolos que incluyen información de direcciones dentro de la conversación del protocolo. [17]"

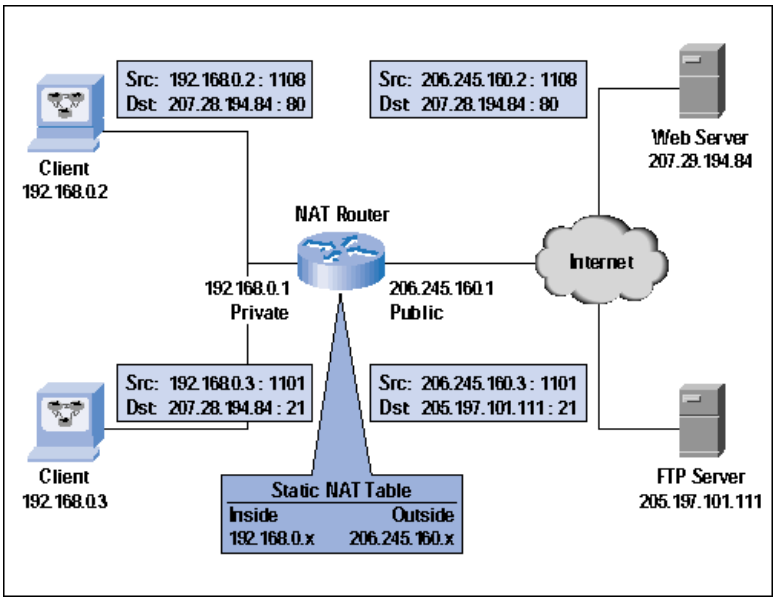


Fig. 56: Nat

Tipos de NAT

Full Cone NAT

Todos los paquetes de la misma dirección y mismo puerto internos son mapeadas a la misma dirección y mismo puerto externo. Cualquier host externo puede mandar un paquete al host interno mandandolo a la dirección y el puerto externo que ha sido mapeado. Se conoce como también como "one-to-one NAT". (NAT uno a uno).

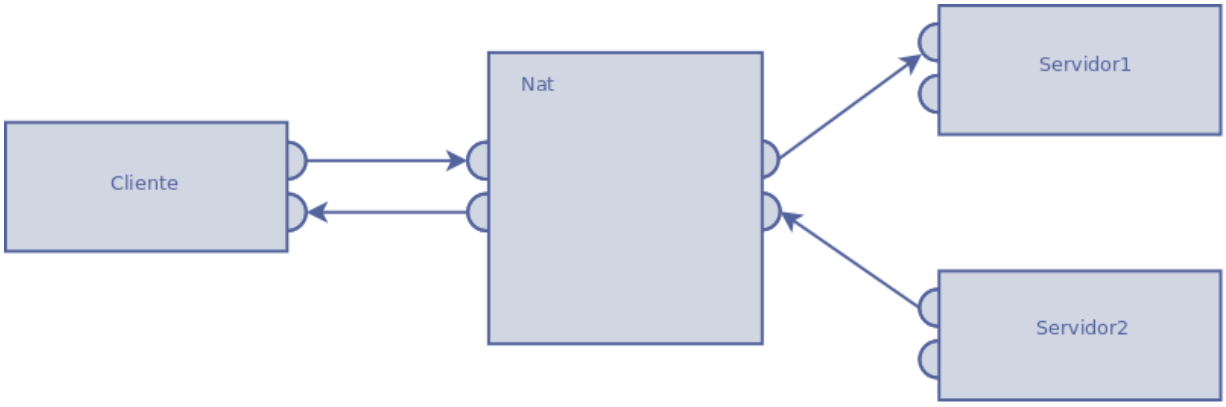


Fig. 57: Full Cone Nat

Restricted cone NAT

Todos los paquetes de la misma dirección y mismo puerto internos son mapeadas a la misma dirección y mismo puerto externo. En este caso, en contraposición con full cone NAT, un host externo (con IP x.x.x.x) sólo puede mandar un paquete al host interno si previamente el host interno le había enviado un paquete a la dirección IP x.x.x.x.

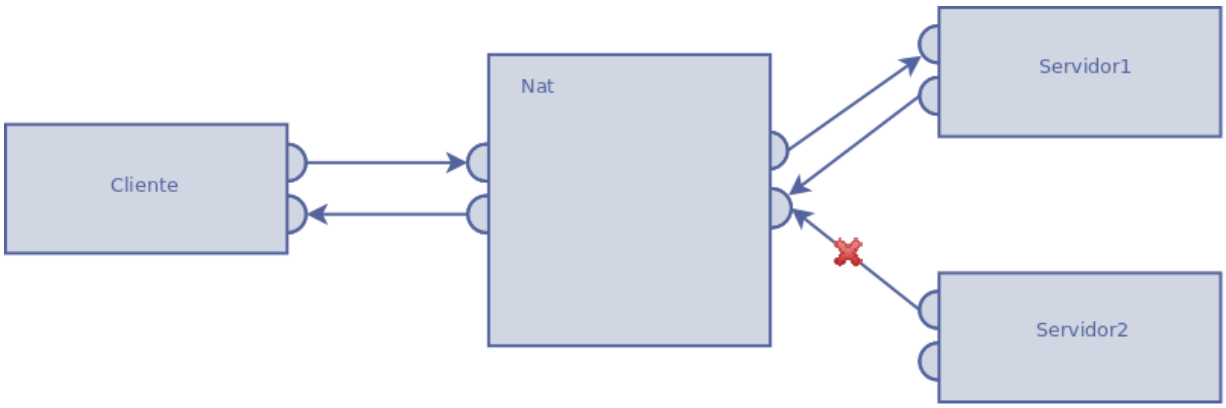


Fig. 58: Restricted Cone Map

Port-restricted cone NAT

Es como restricted cone NAT, pero la restricción incluye también numeros de puerto. Un host externo (con IP x.x.x.x y puerto P) sólo puede mandar un paquete al host interno si previamente el host interno le había enviado un paquete a la dirección IP x.x.x.x y puerto P.

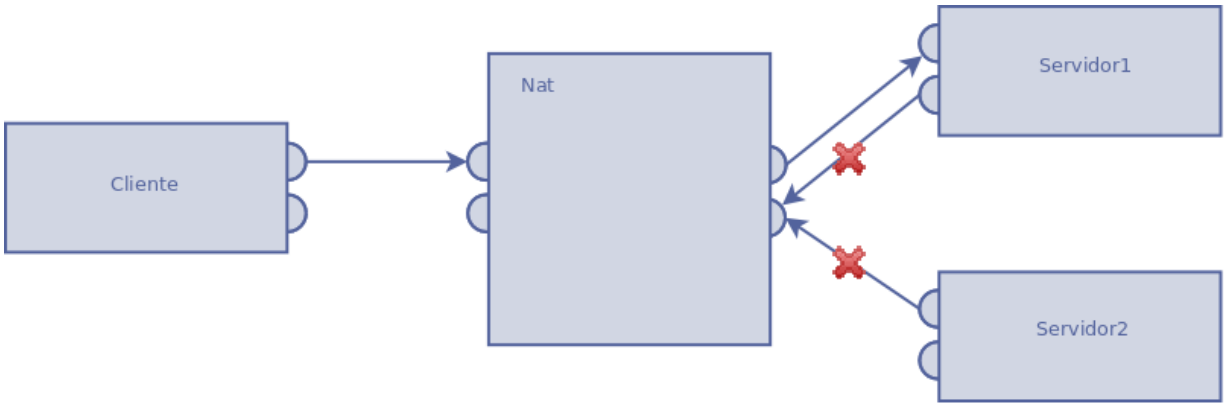


Fig. 59: Cone-Restricted Cone Nat

Symmetric NAT

Es NAT donde todas las peticiones de la misma IP y puerto interno con destino a otra IP y su correspondiente puerto son mapeadas en el router con la misma IP y puerto. Si el mismo host interno manda un paquete con la misma dirección interna y puerto a un destino diferente se usará un mapeo diferente. Sólo el host externo que recibe un paquete puede mandar un paquete UDP de vuelta al host interno.

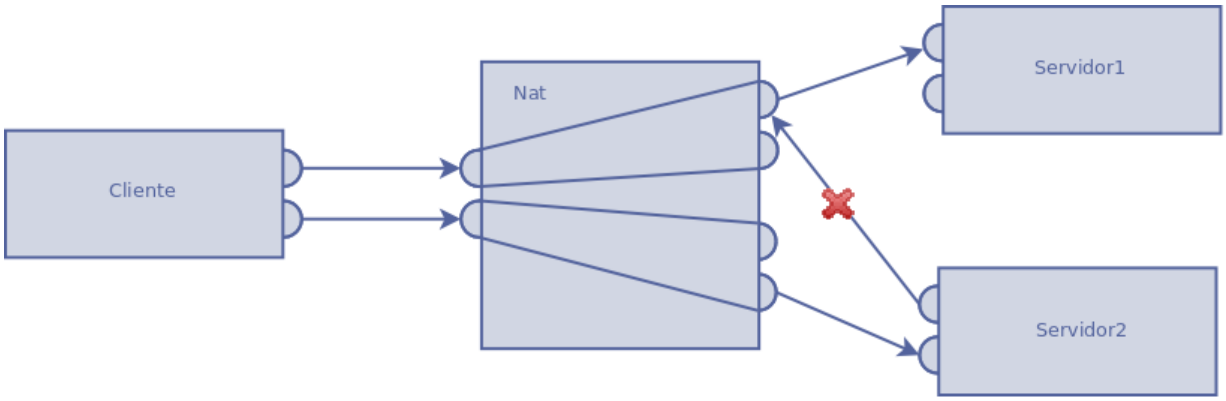


Fig. 60: Symmectric Nat

UDP Hole Punching

La lectura de un artículo [20] sobre la técnica UDP Hole Punching [#punching]_ supuso la inspiración para la realización de este trabajo.

Medienta esta técnica, se pueden conectar, mediante sockets UDP, equipos que están tras firewalles y nats, inaccesibles desde internet directamente, gracias a las características de los elementos de Nat.

"UDP hole punching relies on well-established NAT conventions to allow appropriately designed peer-to-peer applications to "punch holes" through NATs and firewalls and establish direct connectivity with each other, even when both communicating hosts may lie behind a NAT." [20]

Muchas redes usan NAT. Esto permite a sistemas en la misma red compartir una misma IP pública, además de que supone una mejora en la seguridad.

Pero estas ventajas de NAT implica que se produzcan complicaciones a la hora de establecer conexiones P2P.

UDP Hole punching es técnica muy conocida para establecer conexiones P2P entre dispositivos que se encuentra tras NATs. El nombre de "Hole Punching" se debe a que lo que hace es que practica un "agujero" en el firewall de la red que permite que los paquetes provenientes de internet alcancen el equipo destino.

Esta técnica funciona, atravesando cortafuegos y Nats para todos los tipos de NAT excepto para el Nat Simétrico [21] , donde los paquetes de vuelta sólo se permiten de la dirección original inicialmente conectada.

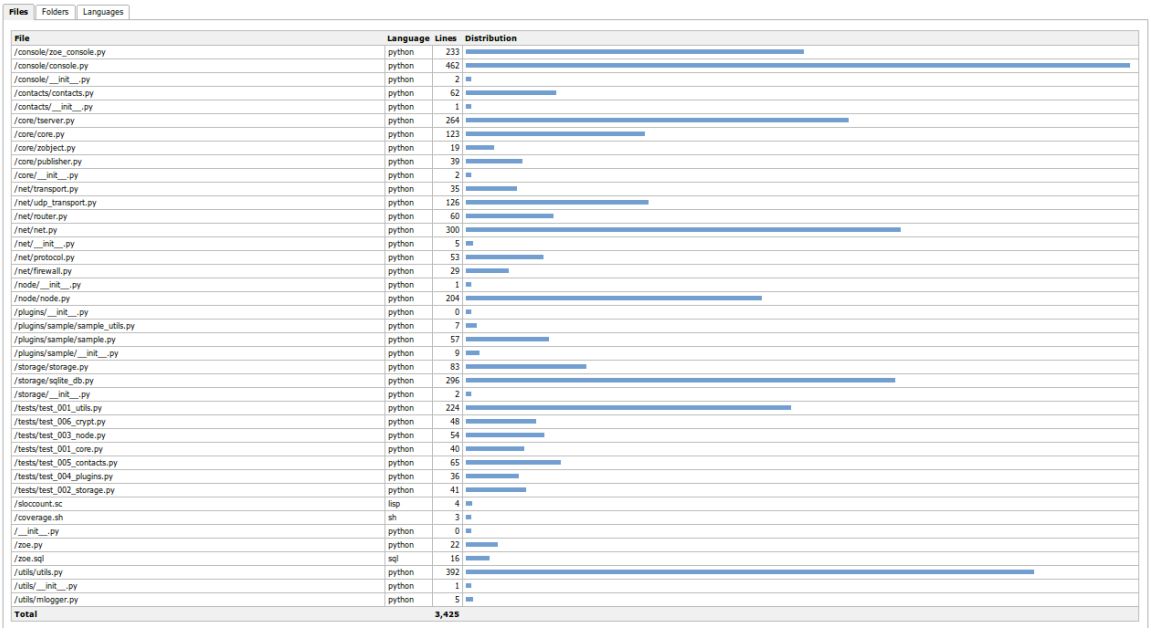
Si bien, en principio, parecía que era imprescindible el uso de un tercer equipo -Presentador-, localizado y accesible en internet, recientes investigaciones apuntan una forma de poder prescindir del tercer equipo [22], si bien la puesta en práctica de lo propuesto en este "paper", queda fuera del alcance de este trabajo y propuesto para futuras ampliaciones o extensiones.

En todo caso, la técnica utilizada para prescindir del presentador requiere que las IPs del nodo a conectar sea conocida por el nodo que establece la conexión. Además, los paquetes ICMP en los que se basa, podrían ser rechazados, por lo que puede no funcionar en todas las ocasiones.

Estadísticas finales

Número de builds: 407

Número de líneas: 3400

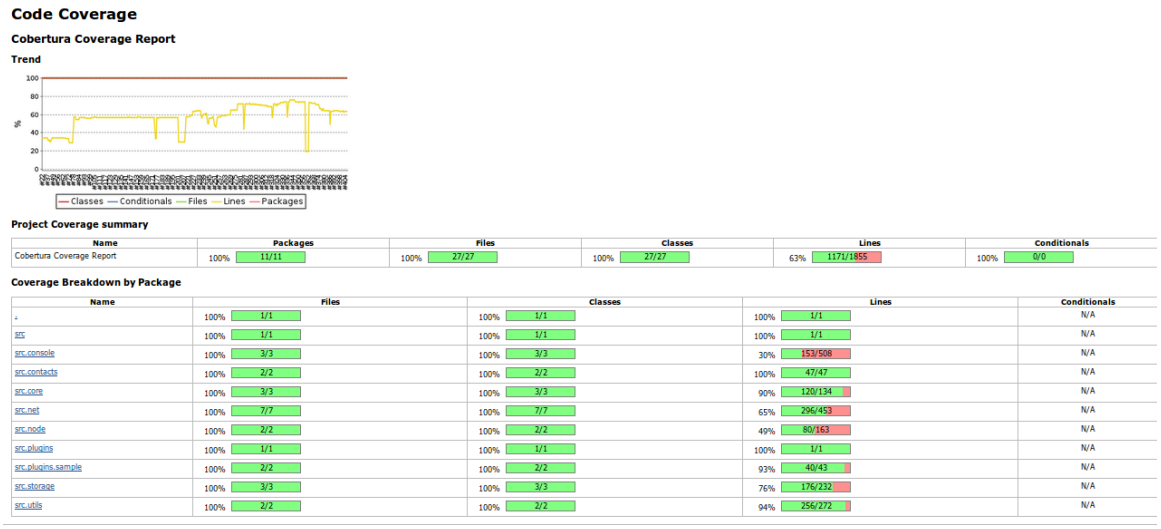


Files | Folders | Languages

File	Language	Lines	Distribution
/console/zoe_console.py	python	233	
/console/console.py	python	462	
/console/__init__.py	python	2	
/contacts/contacts.py	python	62	
/contacts/__init__.py	python	1	
/core/tserver.py	python	264	
/core/core.py	python	123	
/core/zobject.py	python	19	
/core/publisher.py	python	39	
/core/__init__.py	python	2	
/net/transport.py	python	35	
/net/udp_transport.py	python	126	
/net/router.py	python	60	
/net/net.py	python	300	
/net/__init__.py	python	5	
/net/protocol.py	python	53	
/net/firewall.py	python	29	
/node/__init__.py	python	1	
/node/node.py	python	204	
/plugins/__init__.py	python	0	
/plugins/sample/sample_utils.py	python	7	
/plugins/sample/sample.py	python	57	
/plugins/sample/__init__.py	python	9	
/storage/storage.py	python	83	
/storage/sqlite_db.py	python	296	
/storage/__init__.py	python	2	
/tests/test_001_utils.py	python	224	
/tests/test_006_crypt.py	python	48	
/tests/test_003_node.py	python	54	
/tests/test_001_core.py	python	40	
/tests/test_005_contacts.py	python	65	
/tests/test_004_plugins.py	python	36	
/tests/test_002_storage.py	python	41	
/sloccount.sc	lisp	4	
/coverage.sh	sh	3	
/__init__.py	python	0	
/zoe.py	python	22	
/zoe.sql	sql	16	
/utils/utils.py	python	392	
/utils/__init__.py	python	1	
/utils/mlogger.py	python	5	
Total		**3,425**	

Fig. 61: Líneas de código

Grado de cobertura



Code Coverage

Cobertura Coverage Report

Trend

Project Coverage summary

Name	Packages	Files	Classes	Lines	Conditionals
Cobertura Coverage Report	100% 11/11	100% 27/27	100% 27/27	63% 1171/1855	100% 0/0

Coverage Breakdown by Package

Name	Files	Classes	Lines	Conditionals
.	100% 1/1	100% 1/1	100% 1/1	N/A
src	100% 1/1	100% 1/1	100% 1/1	N/A
src.console	100% 3/3	100% 3/3	30% 153/508	N/A
src.contacts	100% 2/2	100% 2/2	100% 47/47	N/A
src.core	100% 3/3	100% 3/3	90% 120/134	N/A
src.net	100% 7/7	100% 7/7	65% 296/453	N/A
src.node	100% 2/2	100% 2/2	49% 80/163	N/A
src.plugins	100% 1/1	100% 1/1	100% 1/1	N/A
src.plugins.sample	100% 2/2	100% 2/2	93% 40/43	N/A
src.storage	100% 3/3	100% 3/3	76% 176/232	N/A
src.utils	100% 2/2	100% 2/2	94% 256/272	N/A

Fig. 62: Cobertura

Violaciones pylint: 9

Problemas conocidos y soluciones propuestas

Sobrecarga

- Problema:

  El protocolo actualmente utilizado se basa en una sencilla serialización del objeto que contiene toda la información del mensaje. Esto genera una importante sobrecarga en las cabeceras.

- Solución propuesta:

  Derivar un protocolo que utilice caberas binarias más eficientes.

- Problema:

  La única forma de encriptación soportada por el momento es una encriptación basada en claves asimétricas RSA donde los mensajes se encriptan con la clave pública del receptor y se firman con la clave privada del emisor. Esto genera una gran sobrecarga para mensajes pequeños.

- Solución propuesta:

  Negociar una clave AES una vez que los nodos son "amigos", utilizando sus claves RSA y encriptar, en adelante, con AES.

Ventana de transmisión

- Problema:

  ZOE es un sistema de mensajes F2F. En la implementación actual, un mensade de A para B sólo puede prosperar si A y B están operativos al mismo tiempo. Esto implica que si no coinciden operativos en algún momento, los mensajes nunca prosperarán.

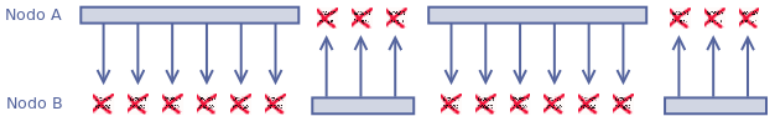


Fig. 63: No hay ventana de transmisión

- Solución propuesta:

  Como solución a esto se propone que se utilicen otros nodos de la red de confianza como colaboradores, de manera que puedan hacer de "puente" entre otros nodos.

  En red de contactos como la siguiente, si cada mensaje se forwardea a otros contactos - encriptado, por supuesto - , además del destinatario legítimo, para que estos intenten enviárselo a su vez, se amplía la posibilidad de tener ventana de transmisión.

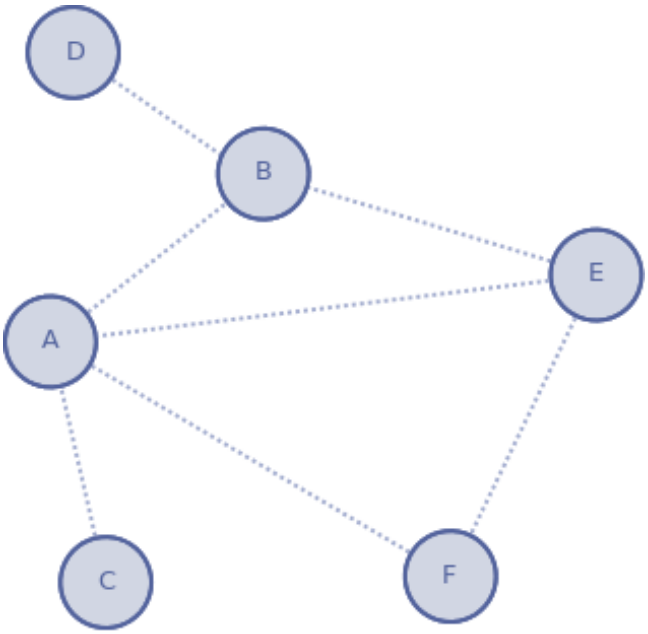


Fig. 64: Red donde los nodos se ayudan a transmitir los mensajes.

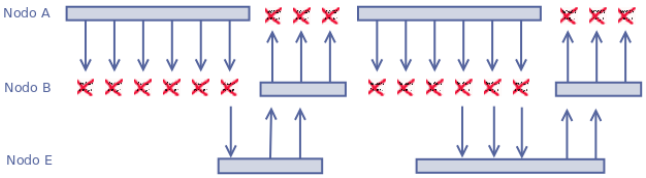


Fig. 65: No hay ventana de transmisión

En este contexto, A enviaría el mensaje para B no sólo a B, sino a todos los nodos que acepten colaborar con él que intentarán, a su vez, hacer llegar el mensaje a B.

Los nodos intermedios no deben poder acceder al contenido de esos mensajes, por lo que deberán ir encriptados.

Se deben implementar estrategias para evitar inundación y bucles.

Para evitar bucles, un nodo no debe procesar un mensaje cuyo **mid** ya ha sido procesado.

Para evitar inundación, los mensajes deben incorporar dos indicadores:

- JTL: Jumps To Live, o cuantos saltos en profundidad pueden alcanzar.
- NTS: Nodes to Spread, o a cuantos nodos se "forwardea" el mensaje.

Consola insegura

- Problema:

  En la implementación actual, la comunicación TCP a través de la consola es plana. Esto, si se utiliza desde el mismo equipo, podría no suponer ningún problema de seguridad, pero si se explota desde otro equipo o, peor, desde internet, la comunicación podría ser comprometida.

  Esto podría resolverse mediante el acceso a través de un canal seguro, como un tunel SSH, pero es algo que queda fuera del alcance del usuario normal.

- Solución propuesta:

  Extender la consola telnet para que soporte conexiones SSL.

Nodos no se conectan tras Nats simétricos

- Problema:

  Udp Hole Punching se basa en que, una vez que un nodo se ha publicado en un Presentador, este comunica su dirección vista en el datagrama (IP:Puerto) a los nodos que quieran conectarse con él.

  Si B está tras un Nat Simétrico, su router sólo admitirá de vuelta paquetes que provengan de la IP del presentador, pero no de terceros.

- Solución propuesta:

  Ampliar la funcionalidad de los nodos para que puedan comportarse como "relays" entre nodos, de la misma manera que hacen los servidores en la red eMule [25] entre nodos que no son accesibles.

  Esto implica que se debe utilizar un servidor Stun [24] para determinar si un nodo está tras un Nat asimétrico y debe utlizarse relay para poder comunicarse con él.

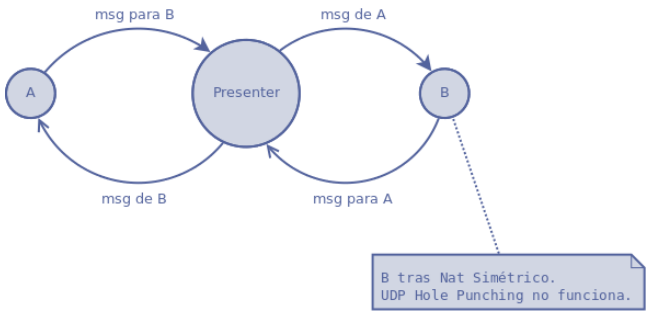


Fig. 66: Comunicacion con nodo tras Nat Simetrico.

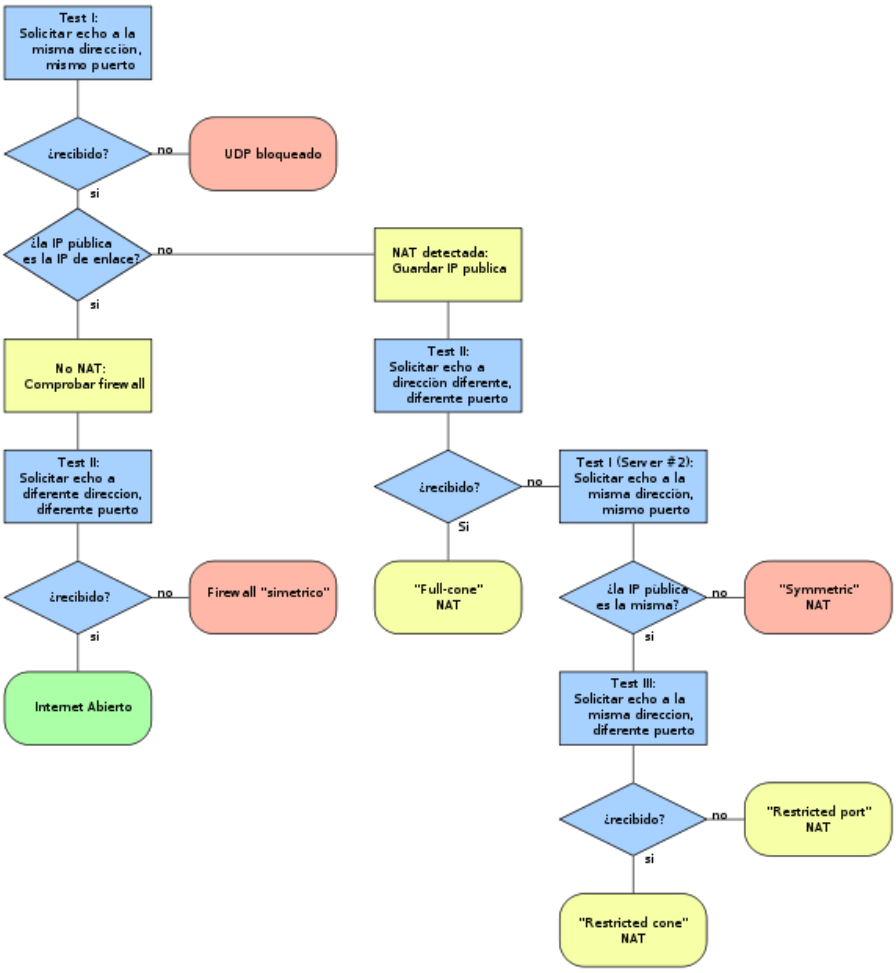


Fig. 67: Protocolo Stun. Fuente: http://es.wikipedia.org/wiki/STUN

Bugs conocidos

Bug.001

- Error: Al parar la aplicación no se detienen todos los procesos
- Reproducibilidad: Baja: Ocurre esporádicamente
- Causa presumible: Algún thread no se detiene

Trabajos futuros

Este trabajo constituye, tan solo, una versión funcional de un "core" que proporciona mensajería encriptada entre dos nodos F2F.

Si bien cumple con su objetivo: "Poder enviar un mensaje encriptado de A a B, sin que este pase por ningún servidor", es necesario, para que sea utilizable en el mundo "real", dotarle de algunas características adicionales.

Algunas de las estas características son:

Instalador

En este trabajo no se ha desarrollado un instalador para usuarios finales.

Eso implica que, actualmente, el usuario deba instalar python en su equipo si no lo tuviera ya y las librarías de python necesarias.

Debería prepararse un instalador típico que con unos sencillos pasos instalaran todo lo necesario para que el nodo fuera operativo.

GUI(s)

Desde el principio, este trabajo se centró en la parte de lógica y se descartó realizar ningún GUI.

En futuros trabajos, podrían escribirse GUIs de usuario, para que sea realmente utilizable por usuarios finales que no están acostumbrados -ni quieren- una austera consola en modo comando.

Dichos GUIs pueden realizarse en cualquier lenguaje explotando el acceso telnet o mediante plugins en python.

Aplicaciones sobre Zoe

ZOE soporta un soporte básico que resuelve la comunicación privada F2F entre nodos.

A partir de aquí se podría extender o usar como core para otras aplicaciones y ofrecer funcionalidades atractivas como:

- Gestión de Grupos
- Presentación de estado y avatar selectivo,
- Compartición y búsqueda de ficheros,
- Backups colaborativos,
- Almacenamiento distribuido,
- Una red social de confianza,
- etc

NetBLT

En la versión actual, por cada mensaje con garantía enviado, se requiere un ACK por parte del receptor y no es eficiente.

El objetivo del presente trabajo no es ser eficiente, pero en futuras ampliaciones o extensiones, sería interesante dotarle de un mejor rendimiento para poder, por ejemplo, intercambiar streamings de vídeo o audio ...

Esto se puede conseguir mediante la utlización de NetBLT [26].

NetBLT (Network Block Transfer) es un protocolo en la capa de transporte diseñado para la transmisión rápida de gran cantidad de información entre ordenadores.

Proporciona una transferencia que es fiable y con control de flujo y está diseñado para proporcionar el máximo throughput sobre una amplia variedad de redes.

Si bien NetBLT actualmente funciona sobre IP, debería ser capaz de funcionar con cualquier protocolo de datagramas.

A grandes rasgos, en NetBLT, todos los mensajes se encolan en un streaming de salida y se envían chunks de datos con cierto número de bloques en cada chunk. Un mensaje no tiene por qué estar completo en un chunk. La comunicación se comporta como un streaming, aunque no lo fuera.

La sobrecarga por acks se minimiza enviando cada cierto tiempo un buffer con máscaras a nivel de bit por cada bloque del chunk. De esta forma, cada extremo envía, cada cierto tiempo, por ejemplo, un buffer de 1Kb con 1024 bits, suponiendo que los chunks fueran de 1024 bloques.

Cada extremo envía entonces sólo los bloques que falten en el otro extremo.

Ref. *NetBLT rfc-998- <http://tools.ietf.org/pdf/rfc998.pdf>*

F.A.Q

Qué significa ZOE ?

Zoe es el nombre de mi hija de tres años.

Por qué python ?

Es cierto que python no es un lenguaje extendido en ambientes empresariales, pero cada vez son más los desarrolladores que apuestan por él debido a sus virtudes, como

- OpenSource:

  Python es un lenguaje OpenSource.

- Productividad:

  El tamaño del código en python para realizar la misma tarea es varias veces menor que en otros lenguajes.

  Al no ser compilado, se evita el ciclo típico de edición/compilación/prueba, pasando directamente de edición a prueba.

- Legibilidad:

  El código de python es muy legible, gracias a su organización tabulada y a la aplicación de PEP8

- Sencillez:

  Python es un lenguaje muy fácil de aprender. El famoso programa "hello world":

  En python:

```
print "hello world"
```

  En C++:

```
#include <iostream>
using namespace std;

int main() {
   cout << "Hola Mundo" << endl;
return 0;
}
```

  En java:

```
public class HelloWorld {

public static void main(String[] args) {
    System.out.println("Hello, World");
}
}
```

- Multiplataforma:
  Al ser un lenguaje interpretado, las aplicaciones python pueden correr en cualquier plataforma que tenga python instalado.

- Modo consola:

  Python proporciona una consola sobre la que se pueden probar rápidamente sus funcionalidades.

  Adicionalmente, mediante una librería, se puede acceder "en caliente" a las entrañas de una aplicación python ejecutándose.

- Librerías:

  Existen cientos si no miles de librerías para python.

- El Zen de python [#zen]_:

  - Bello es mejor que feo.
  - Explícito es mejor que implícito.
  - Simple es mejor que complejo.
  - Complejo es mejor que complicado.
  - Plano es mejor que anidado.
  - Disperso es mejor que denso.
  - La legibilidad cuenta.
  - Los casos especiales no son tan especiales como para quebrantar las reglas.
  - Aunque lo práctico gana a la pureza.
  - Los errores nunca deberían dejarse pasar silenciosamente.
  - A menos que hayan sido silenciados explícitamente.
  - Frente a la ambigüedad, rechaza la tentación de adivinar.
  - Debería haber una -y preferiblemente sólo una- manera obvia de hacerlo.
  - Aunque esa manera puede no ser obvia al principio a menos que usted sea holandés.15
  - Ahora es mejor que nunca.
  - Aunque nunca es a menudo mejor que ya mismo.
  - Si la implementación es difícil de explicar, es una mala idea.
  - Si la implementación es fácil de explicar, puede que sea una buena idea.
  - Los espacios de nombres (namespaces) son una gran idea ¡Hagamos más de esas cosas!

Por qué no se ha incluido un GUI ?

El objetivo de este trabajo es diseñar y desarrollar un "core" completamente operativo que sea capaz de intercambiar mensajes entre peers de forma directa (sin que la información pase por servidores) y encriptada.

Como criterio de diseño, el core dispone, a la manera de mldonlkey, una consola telnet y un sistema de plugins que permite que el GUI esté completamente desacoplado del mismo y que pueda ser desarrollado por terceros de la forma que más convenga. De hecho, se pueden desarrollar múltiples GUIs con diferentes tecnologías.

Por qué no se ha implementado para dispositivos móviles como Android o iPhone ?

No es objetivo de este trabajo realizar una aplicación para android o iPhone, sino diseñar, desarrollar y validar una tecnología de comunicaciones.

En este sentido, linux es, a juicio del autor, el mejor "ambiente" para ello.

No obstante, se ha evitado utilizar funcionalidades específicas de linux y en el código escrito no se han utilizado facilidades propias de python como funciones lambda [27] o generadores yield [28], de manera que el código sea fácilmente portable a cualquier plataforma.

Cómo se puede evitar la sobrecarga en las comunicaciones ?

Existen dos motivos fundamentales para que ZOE añada una gran sobrecarga a las comunicaciones:

- Por una parte, el protocolo de prueba suministrado -Zoe- se limita a hacer una serialización del mensaje enviado y los datos necesarios para la cabecera del protocolo.

  Es muy sencillo derivar otro protocolo más eficiente que sea binario.

- Por otra parte, la encriptación fuerte RSA de 512 bits y la firma de los mensajes hace que los datagramas enviados sean mucho mayores que el payload.

  Posibles soluciones pasan por, o bien no encriptar si no es necesario o bien negociar una clave AES, encriptada con RSA y, a partir de ahí, utilizar esa clave AES.

Es necesario que exista un Presentador ?

El presentador es necesario siempre que se necesiten traspasar NATs. Independientemente de que un nodo especifique cual ha de ser su puerto UDP, ese puerto será distinto una vez que sus mensajes salgan a internet.

La IP pública se podría conocer de otras formas, pero la única forma de conocer el puerto UDP por el que se sale es que exista un nodo -el presentador- que tenga un puerto UDP accesible y así obtenga la dirección física.

Además, el presentador es el que pone en conocimiento a dos nodos que se quieren conectar, enviándoles un pequeño datagrama a ambos con información de contacto del otro extremo.

En cualquier caso, todos los nodos de ZOE son presentadores en potencia. Sólo es necesario que utilicen un puerto UDP conocido y accesoble.

Y no es, entonces, un SPOF el Presentador?

En la implementación actual es imprescindible un presentador para que los nodos puedan localizarse. Pero si se tiene en cuenta que todos los nodos son presentadores si sus puertos udp fueran accesibles, resulta fácil poder desplegar una red de confianza donde siempre existan presentadores "vivos".

En ambiente LAN se podría utilizar una técnica de broadcast para localizar otros nodos de ZOE en la red local, sin dependencia de presentador, pero no es del alcance de este trabajo.

En ambiente internet, los presentadores se podrían publicar, por ejemplo, en una red DHT o en un cluster de "servidores de presentadores" de forma que se garantizara la localización de ellos.

Si existiera un sólo presentador conocido, ciertamente sería un SPOF.

Cual es el modelo de negocio de ZOE?

Pese a que este trabajo comenzaba hablando sobre la batalla entre las empresas por dominar el mercado de los mensajes OTT, la vocación de Zoe no es ser una solución empresarial. Está concebido para que la gente pueda comunicarse de forma privada entre ellos, sin que sus mensajes y contenidos pasen por ningún servidor susceptible de ser espiado o comprometido.

Pero se puede hacer: si se quisiera establecer algún modelo de negocio sobre ZOE, podría pasar por desplegar Presentadores que ofrecieran funcionalidades atractivas a los usuarios. En este contexto, el modelo de negocio podría consistir en cobrar cuotas por servicios o tener ingresos por publicidad.

ZOE sirve para localizar contenidos otros nodos unidos a la red ?

No por defecto. Zoe proporciona un core que es capaz de enviar y recibir mensajes a contactos de confianza, sin necesidad de conocer a priori sus direcciones, traspasando NATs y cortafuegos y de forma encriptada.

Pero puede ser extendido fácilmente para localizar y compartir contenido expuesto por los contactos.

Si la comunicación es P2P, qué pasa si dos nodos no coinciden conectados en la misma ventana de tiempo ?

Los mensajes no llegarán jamás.

Esto puede resolverse fácilmente en futuras ampliaciones de este trabajo, de manera que los contactos elegidos puedan ser "rely" de mensajes que no son para ellos. De esta forma se incrementa la posibilidad de que los mensajes prosperen.

Por qué no se intercambian las claves públicas en la invitación/aceptación ?

Un ataque "Man in the middle" [#man_middle] podría interceptar las comunicaciones y proporcionar su propia clave pública. Ese atacante podría interceptar las comunicaciones posteriormente encriptadas y pasarlas de un extremo a otro sin que ninguno se percatara de ello.

Por eso, las claves públicas se deben intercambiar por algún método seguro garantizado.

Qué hace falta para empezar a usar ZOE ?

- Deben existir, al menos, dos nodos y
- Tener python instalado.
- Copiar los ficheros de ZOE en el equipo a utilizar.
- Editar el fichero config.cfg e indicar:
    - id (normalmente una dirección de correo propio)
    - Puerto UDP si es que está accesible -bien directamente o porque se ha podido hacer un NAT en el router)
    - IP:puerto del presentador (al menos uno de los dos nodos debe hacer de presentador)
- Arrancar zoe: ./zoe.py
- En tanto existan GUIs, utilizar la consola telnet para operar.

Note

Puesto que, en principio, no existen Autoridades Presentadoras, la única forma de garantizar la privacidad en las comunicaciones es enviando por un canal seguro (en mano, email, otros ...) nuestra clave publica <email>.pub a los contactos que vayamos invitar o por los que seamos invitados.

Note

Como futuras ampliaciones de este trabajo, se podría implementar un instalador que minimice las tareas de configuración

---

1	http://docs.python.org/release/2.5/lib/module-cPickle.html
2(1, 2, 3)	http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER09/hadad.pdf
3	http://www.eclipse.org/
4	http://www.gnuplot.info/
5	https://projects.gnome.org/dia/
6	http://es.wikipedia.org/wiki/Integraci%C3%B3n_continua
7	http://jenkins-ci.org/
8	http://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software
9	https://www.scrum.org/
10	http://www.blackwasp.co.uk/Facade.aspx
11	http://es.davidhorat.com/publicaciones/articulos/patrones/observador/
12	*Man in the middle attack <http://es.wikipedia.org/wiki/Ataque_Man-in-the-middle>*
13	*RSA rfc-3447 <http://tools.ietf.org/pdf/rfc3447.pdf>*
14	*IPv4 <rfc-791 http://tools.ietf.org/pdf/rfc791.pdf>*
15	*IPv6 <frc-2460 http://tools.ietf.org/pdf/rfc2460.pdf>*
16	*Nat rfc-4787 <http://tools.ietf.org/pdf/rfc4787.pdf>*
17	http://es.wikipedia.org/wiki/Network_Address_Translation
18	http://www.monografias.com/trabajos20/traductor-nat/traductor-nat.shtml
19	Peer to Peer communication across Nework Address Translators`<http://pdos.csail.mit.edu/papers/p2pnat.pdf>`
20(1, 2)	http://pdos.csail.mit.edu/~baford/nat/draft-ford-natp2p-00.txt
21	http://en.wikipedia.org/wiki/Network_address_translation#Types_of_NAT
22	http://resources.infosecinstitute.com/udp-hole-punching/
23	http://grothoff.org/christian/pwnat.pdf
24	*Stun RFC-5389 <http://tools.ietf.org/pdf/rfc5389.pdf>*
25	*eMule Protocol <http://www.cs.huji.ac.il/labs/danss/p2p/resources/emule.pdf>*
26	*NetBLT rfc-998- <http://tools.ietf.org/pdf/rfc998.pdf>*
27(1, 2)	lambda en python
28(1, 2)	yield en python