TECNOLOGÍAS DE LA INFORMACIÓN AL SERVICIO DE LA HISTORIA CLÍNICA ELECTRÓNICA

Pere Crespo Molina
José Alberto Maldonado Segura
Montserrat Robles Viejo
Área de Informática Médica del Grupo BET de la Universidad Politécnica de Valencia

Miguel Chavarría Díaz
Hospital la Fe. Departamento de Informática

Este capítulo tiene el propósito de describir cómo puede encontrarse almacenada la información clínica, ya sea en un sistema centralizado o distribuido y los problemas o ventajas que esto conlleva. En el sector sanitario predominan los sistemas distribuidos que son, además, heterogéneos y autónomos entre sí. También existen multitud de sistemas obsoletos y que son difíciles de mantener. Por este motivo, en este capítulo también se tratan los temas de integración de la información distribuida y migración a sistemas más modernos. Además, en este capítulo se ofrece una introducción a cuatro tecnologías de amplia actualidad que pueden ayudar en el desarrollo de nuevos sistemas de historias clínicas o en la integración de los ya existentes: XML como formato para la representación de la información clínica, ontologías para describir formalmente la información y el conocimiento y dos tipos de middleware: CORBA y los web services, que facilitan la interconexión de aplicaciones distribuidas. Por último, se dan unas breves pinceladas sobre cómo debe presentarse la información a los usuarios finales.

SISTEMAS DE INFORMACIÓN: CENTRALIZADO Y DISTRIBUIDO

Sistemas centralizados

Desde el inicio de la era de la computadora moderna, 1945, hasta cerca de 1985, solo se conocía el paradigma de arquitectura de información centralizada. De hecho, a principios de los 80 el procesamiento de la información de manera computarizada estaba estrictamente centralizado. Las principales características de esta centralización eran las siguientes:

a) Uso de una computadora principal (*mainframe*). Normalmente un gran ordenador (IBM System 9000) o una minicomputadora (VAX). En términos de estructura de la compañía, una unidad era responsable del mantenimiento de dicha computadora mientras sus recursos estaban disponibles para el resto del personal. A esta unidad se la llamaba unidad de informática o centro de cálculo.

b) Procesamiento centralizado de la información. Todos los procesos y cálculos son ejecutados por el ordenador principal. Los diferentes departamentos tienen terminales conectados a este ordenador (*mainframe*).

c) Centralización de la información. La información es almacenada en dispositivos de almacenamiento bajo el control del *mainframe*.

d) Control centralizado. El administrador del sistema es el único responsable del procesamiento de la información del sistema. El administrador autoriza el acceso de los usuarios, es responsable del funcionamiento, soporte y seguridad diaria del sistema.

e) Servicio centralizado. El hardware y el software están mantenidos por el personal del centro de informática. Los otros departamentos no tienen técnicos informáticos.

La reserva de vuelos aéreos es un ejemplo típico de sistema centralizado. Hay un único ordenador central que sirve a un gran número de puntos de venta. Mantiene información acerca de vuelos y vacantes. Toda la reserva de tickets se lleva a cabo en el ordenador central, mientras que las terminales solo son puntos de venta desde los cuales se introduce información de reserva y se imprimen los resultados cuando se confirman éstas.

Entre las principales ventajas de los sistemas centralizados podemos citar las siguientes:

– Fáciles de manejar.
– Implican menos costes de personal.
– Un ordenador es suficiente para soportar diferentes sistemas de información en una única compañía.
– Asignación de nombres uniforme (de usuarios, de ficheros, de servicios).
– La localización de los objetos no es un problema.
– El acceso y la seguridad se controlan uniformemente en todo el sistema.
– La gestión y administración puede hacerse de forma centralizada.

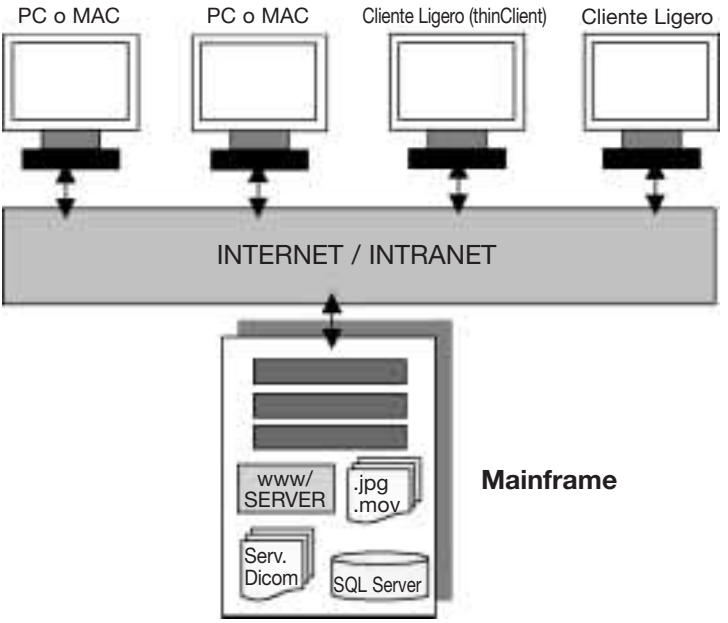
En resumen, los sistemas centralizados se caracterizan por su:

– Accesibilidad. Cualquiera puede usar todos los recursos.
– Coherencia. Todo funciona de igual forma en todas partes.
– Gestión centralizada. Único dominio de gestión.

Actualmente, rara vez se encuentra una situación en la que sea justificable una centralización del estilo de lo descrito anteriormente. Solo tiene sentido hablar de centralización a nivel de datos o de servicios cuando por su naturaleza es mejor tenerlos centralizados. De todas formas, ahora es bastante habitual tener lo que se

ha venido en llamar clientes ligeros o "*thin clients*" en inglés, que son terminales de bajo coste, sin disco duro y otros recursos, pero con capacidades multimedia y que están conectados a un ordenador central más potente. En algunas situaciones en las que se desea que un ordenador se utilice para un conjunto acotado de aplicaciones puede ser una opción económica e interesante a tener en cuenta. La figura 1 muestra un esquema general de un sistema centralizado.

Figura 1. Sistema centralizado



En sanidad, un ejemplo de centralización de un servicio importante podría ser la función de asignación de un identificador único de paciente. Si estuviese centralizada esta información se evitarían algunos escenarios bastante comunes e indeseables en algunos centros hospitalarios como son que un mismo paciente tenga diferentes identificadores en los diversos sistemas de la organización. Tal vez no es estrictamente necesario que para mantener un registro único de identificadores de paciente esta información deba estar centralizada, podría estar distribuida entre varios sistemas pero de alguna manera el control de estos debería estar centralizado. Conseguir un sistema estrictamente distribuido sería más complejo aunque

existen ya numerosas implementaciones comerciales totalmente distribuidas basadas en las recomendaciones que CORBAmed sugiere sobre este tema.

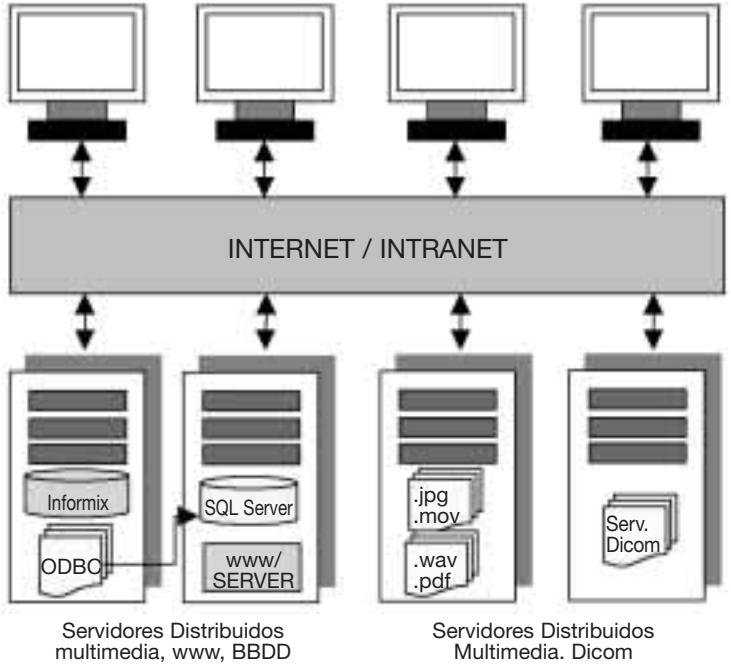
Sistemas distribuidos

La filosofía de un sistema distribuido es la opuesta a la de uno centralizado. Un sistema distribuido se define como un conjunto interconectado de entidades autónomas. Dichas entidades pueden ser:

– Computadoras
– Procesadores
– Procesos

Actualmente, el escenario típico de cualquier organización sanitaria es un panorama en el cual coexisten numerosos sistemas de información distribuidos y heterogéneos repartidos por cada uno de los departamentos y que rara vez son interoperables con el resto de sistemas, constituyendo así una organización con numerosas islas de información. La Figura 2 contiene un ejemplo de sistema distribuido.

Figura 2. Sistema Distribuido



Con este problema se evidencia uno de los principales inconvenientes de los sistemas distribuidos, si los sistemas de información no crecen de manera ordenada y bajo unos criterios mínimos de compatibilidad e interoperabilidad, acaban convirtiéndose en sistemas aislados que realizan muy bien su tarea en el departamento en el que fueron implantados pero no pueden interoperar con nadie y además abocan a situaciones de redundancia de información, inconsistencia, dificultad en la gestión de indicadores, cálculos estadísticos etc. de la totalidad de una organización, etc. Lamentablemente esta situación obliga a la mayoría a recurrir a soluciones para integrar estos sistemas para minimizar en parte estos efectos.

El contexto tecnológico actual obliga a la distribución de los sistemas pero hay que tener mucho cuidado en su evolución ya que no se puede controlar todo de manera centralizada como se hacía antes. Si no se sigue una estricta política en cuestiones de compatibilidad, estándares, escalabilidad, tecnología etc. entre los diferentes sistemas de la organización sanitaria podemos llegar a escenarios indeseables como los anteriormente descritos.

La promesa de una mayor robustez, apertura e infinita escalabilidad de los nuevos sistemas distribuidos ha hecho que en los entornos sanitarios, mayoritariamente, se apueste por la descentralización o simplemente de manera inconsciente y natural se ha tendido hacia ella. Por esta razón, posiblemente sea necesario abordar en dichas organizaciones el diseño de un sistema de información distribuido.

La amplia difusión de los sistemas distribuidos es el resultado de los siguientes logros en el mundo de la tecnología:

– Descenso de los precios de los ordenadores personales y amplia aplicación de los ordenadores en casi todos los ámbitos. El precio de los ordenadores personales ha descendido enormemente, esto es más llamativo aún en el caso de los servidores de red local. Además estos ordenadores son más potentes que los grandes *mainframes* de los años 80.

– Desarrollo de redes de ordenadores. El desarrollo de las comunicaciones ha hecho posible la interconexión de ordenadores en redes de área local (LAN) así como en redes de área extensa (WAN). Una red de una organización es un ejemplo de LAN e Internet un ejemplo de WAN.

La amplia difusión de los PCs y las mejoras en la comunicación de ordenadores han promovido el concepto de trabajo en grupo. Esto ha sido el inicio de los sistemas distribuidos. Las principales características son:

– Ordenadores distribuidos. El sistema está compuesto por un cierto número de ordenadores instalados en diferentes localizaciones e interconectados entre sí.

– Proceso de la información distribuido. La información, procesada por un sistema puede estar distribuida entre muchos ordenadores y quedar accesible a cualquiera de ellos.

– Control distribuido. Los sistemas distribuidos normalmente no están bajo un control centralizado. Los ordenadores del sistema están bajo control de los usuarios que están trabajando en esos momentos con ellos. Algunas veces, el control distribuido se combina con uno centralizado. Por ejemplo, es deseable controlar el entorno de red y su estado de una manera centralizada.

Las principales ventajas de los sistemas distribuidos son las siguientes:

– Rapidez de respuesta. Los recursos de los ordenadores en un sistema distribuido son compartidos, de modo que el sistema puede procesar cada petición de usuario de la manera más rápida posible.

– Rendimiento. La operación combinada de muchos ordenadores mejora el rendimiento.

– Fiabilidad. En el caso de los sistemas centralizados, si el ordenador principal falla el sistema en su globalidad no se puede usar. No se puede ofrecer ningún servicio hasta que se repare. Si un ordenador en un sistema distribuido falla, la carga puede ser distribuida entre el resto, continuando así el funcionamiento del sistema aunque sea a costa de un rendimiento más bajo.

– Escalabilidad. Un sistema distribuido puede crecer más fácilmente que uno centralizado. Si una compañía, que utiliza un sistema centralizado, decide crecer y mejorar la productividad de manera significativa, necesita adquirir un nuevo ordenador central. En cambio si el sistema es distribuido, la productividad se puede incrementar solamente adquiriendo nuevos ordenadores personales o *workstations*.

– Consistencia con la estructura de la organización. Es necesario relacionar el sistema con la estructura organizativa. Por ejemplo, es natural que los ordenadores encargados del sistema de radiodiagnóstico se encarguen de todo lo relacionado con este servicio, mientras que el departamento de personal se encarga de las tareas de gestión administrativa.

– Participación del usuario en el desarrollo del sistema. Un sistema distribuido facilita que un usuario tenga mejores oportunidades de participar en su diseño y mejora. Los usuarios de los sistemas distribuidos no solo usan los servicios centralizados sino que también llevan a cabo tareas independientes.

– Flexibilidad. Un sistema distribuido se puede afinar fácilmente para que se ajuste a los cambios de los requerimientos de los usuarios. Como un grupo de

usuarios solo opera con un subsistema es posible su modificación sin que esto afecte a los usuarios que interaccionan con otros subsistemas.

La discusión sobre las ventajas y desventajas de los sistemas distribuidos frente a los centralizados es un punto extenso y muchas veces la orientación de la solución idónea puede no ajustarse a los intereses de la organización. De todas formas es utópico hablar de un tipo u otro de organización de los sistemas de información en términos absolutos ya que no existen de manera pura en casi ningún entorno, presentándose frecuentemente situaciones híbridas.

Sin lugar a dudas, los sistemas distribuidos son el camino considerablemente más avanzado para los servicios de información de una organización, aunque revelan también una serie de puntos flacos entre los que podemos citar los siguientes:

– Aumento de la complicación en el diagnóstico de fallos. Normalmente detectar el componente que origina un fallo en todo el sistema distribuido es más complicado.

– Incompatibilidad de la información, islas de información. Si los diferentes componentes del sistema han sido diseñados independientemente, lo que es frecuente para los sistemas distribuidos, entonces la información producida por éstos puede ser incompatible con otros subsistemas.

– Protección y disponibilidad. La existencia de numerosos sistemas pueden ocasionar que la protección del sistema en su conjunto sea parcial y que alguno de ellos en un momento dado no este disponible. Este tema se aborda con más detalle en el capítulo *Seguridad, confidencialidad y disponibilidad de la información clínica* de este libro.

Existen diferentes arquitecturas de los sistemas distribuidos de las cuales haremos un repaso a las más conocidas y utilizadas a lo largo de los últimos años.

1. Arquitectura cliente-servidor

Los sistemas distribuidos más extendidos son de este tipo. Una parte de estos sistemas (los servidores) proveen operaciones comunes llamadas servicios, mientras otros (los clientes) acceden a estos servicios. El *e-mail* es un típico ejemplo de la arquitectura cliente-servidor. Los usuarios utilizan aplicaciones cliente para escribir y leer su correo electrónico. Mientras tanto, un servidor de correo se está ejecutando y recibe el flujo de e-mails del exterior y distribuye éstos entre los diferentes buzones de los usuarios (cuentas). A su vez, también se encarga de encaminar los correos que escriben los clientes hacia otros servidores donde están ubicados los buzones de los destinatarios. FTP, WWW y todos los servicios asociados a Internet suelen seguir la arquitectura cliente-servidor.

El modelo cliente-servidor es conveniente cuando lo que se quiere es separar tareas comunes y especializadas (servidores) de aquellas que son individuales (clientes). Estas tareas están caracterizadas por diferentes niveles de soporte. Las tareas cliente son simples y se comunican directamente con el usuario. Las tareas de servidor son más complejas, llevan a cabo un sinfín de complejos servicios que normalmente requieren de personal especializado para su mantenimiento técnico.

De acuerdo con su estructura, los sistemas distribuidos también se suelen diferenciar habitualmente en la literatura entre verticales y horizontales.

– Sistemas distribuidos verticales. Las funciones que procesan la información de un sistema vertical están en sintonía con las tareas o con la estructura jerárquica de la organización. Un sistema de gestión de seguros puede servir como ejemplo. Puede ser descrito como un sistema jerárquico de dos niveles. El sistema superior, correspondiente a la oficina central, lleva a cabo tareas de análisis de riesgos, cálculo de tasas y otras operaciones centralizadas. El sistema inferior, correspondiente a las diferentes sucursales, se encarga de los contratos de pólizas. Finalmente la información de las pólizas es enviada regularmente desde las sucursales a la oficina central.

– Sistemas distribuidos horizontales. Las funciones que procesan la información están distribuidas entre diferentes puestos de trabajo equivalentes. Estos pueden intercambiar información entre ellos para asegurar la funcionalidad del sistema. Una oficina automatizada puede ser un buen ejemplo. Los oficinistas utilizan el ordenador para ofimática, contabilidad etc. Los ordenadores están interconectados. Se pueden intercambiar archivos, mensajes y otro tipo de información sin tener un nivel jerárquicamente superior.

2. Arquitectura multicapa o n-capas

Son una evolución/refinamiento del modelo cliente-servidor. Es la apuesta de plataformas tecnológicas como son J2EE (Java) y .NET.

La arquitectura es la estructura organizativa de un sistema, que incluye su descomposición en partes, conectividad, mecanismos de interacción y principios de guía que proporcionan información sobre el diseño del mismo. Actualmente se definen tres estilos de arquitecturas para las aplicaciones distribuidas:

– Arquitectura de 2 capas

– Arquitectura de 3 capas

– Arquitectura de n capas

El propósito principal es fragmentar las aplicaciones en una serie de piezas o bloques funcionalmente similares, desacoplados y con cierto grado de independen-

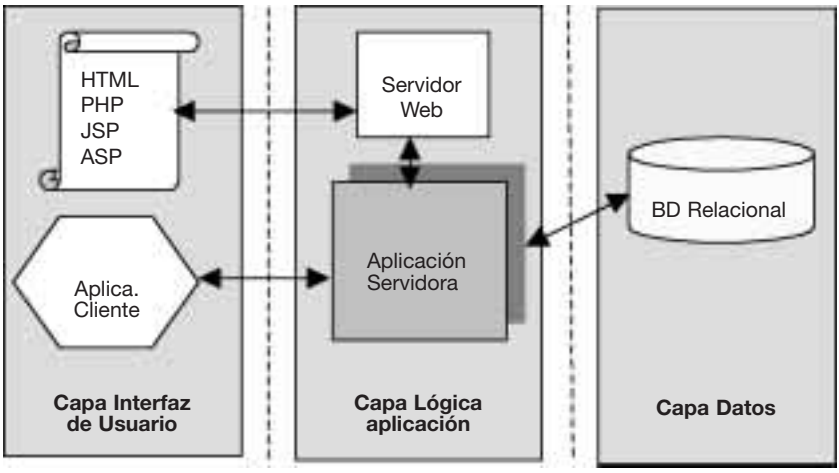
cia respecto a las demás. Con esto, generalmente, se consigue un sistema con un balanceo de carga más equilibrado entre clientes y servidores. Además, aunque un sistema multicapa es, en general, más complejo que uno de 2 capas (cliente-servidor clásico) se mejora el mantenimiento y la escalabilidad del sistema una vez puesto en producción ya que cada parte puede evolucionar o cambiarse sin que esto afecte al resto. De hecho, en un sistema de 3 capas ya se puede apreciar esta característica, vease Figura 3.

Puede cambiarse la interfaz de usuario de la aplicación, por ejemplo, para mejorarla o para que sea una interfaz vía Web o cualquier otra, sin que esto afecte para nada al resto de las capas.

.Net y J2EE proponen una colección de capas para sus aplicaciones empresariales, para que el trabajo distribuido en una organización junto con sus necesidades de información sea gestionado por servidores de aplicaciones que abstraen y resuelven a los programadores tareas de comunicación, transparencia de acceso a datos, transacciones, persistencia, etc.

Uno de los inconvenientes principales de los sistemas multicapa es el tiempo de desarrollo y curva de aprendizaje inicial que suele ser más elevado que el de una aplicación cliente-servidor o convencional. También es habitual que al tener una aplicación con muchas capas y componentes, a veces de terceras partes, hace que los niveles de indirección se incrementen de tal manera que la depuración y testeo de errores sea una quimera complicada.

Figura 3. Arquitectura de 3 capas



3. Grid Computing

Surge por la amplia difusión de Internet y de la Web como nuevo paradigma de la computación distribuida a gran escala. La computación GRID es un tipo de sistema paralelo y distribuido que permite la compartición, selección y agregación de cada uno de los recursos computacionales distribuidos en una área geográfica extensa (supercomputadoras, clusters de computación, sistemas de almacenamiento, fuentes de datos, instrumentos, gente) y presenta todos estos recursos como uno solo, unificándolos para solucionar la computación a gran escala y el uso intensivo de información por parte de las aplicaciones como, por ejemplo, el modelado molecular para el diseño de medicamentos, el análisis de la actividad cerebral, etc.

Esta idea es análoga a la red eléctrica, dónde los generadores están repartidos por todo el territorio pero los usuarios acceden a la electricidad sin preocuparse de cuál es la fuente y cuál es su localización.

Los Web Services, a los cuales se dedica más adelante un apartado en este capítulo, tienen puntos en común con Grid pero no son exactamente lo mismo, más bien los Web Services son una de las posibilidades de Grid. El concepto de servicio en Grid incorpora una serie de características muy interesantes de las que carecen los Web Services, entre las que se destacan:

– *Statefulness* (SDEs) (con estado)

– Interacciones *Stateful* (con estado)

– Instancias transitorias (Factorías)

– Gestión del ciclo de vida

– Introspección

– Notificación de cambios de estado

INTEGRACIÓN DE LA HISTORIA CLÍNICA ELECTRÓNICA

Concepto de integración de datos

Cada día más, la atención sanitaria de un paciente es la responsabilidad compartida de un grupo de profesionales pertenecientes a diversas disciplinas o instituciones. Como consecuencia de ello, es vital que las instituciones sanitarias puedan compartir información sobre los pacientes, de una manera sencilla, segura y con-

servando el significado original de los datos. Pero actualmente esta información se encuentra repartida en múltiples sistemas de información heterogéneos y autónomos, lo que hace que el acceso uniforme a los registros clínicos sea una tarea problemática.

La integración de datos es el problema de combinar datos que residen en distintos sistemas, posiblemente heterogéneos entre sí y proporcionar a los usuarios finales una vista unificada de estos datos. En el caso de los sistemas de información sanitarios que contiene información clínica sobre los pacientes, la integración tiene, generalmente, como propósito el facilitar a los usuarios (ya sea personal clínico, administrativo, de investigación o el propio paciente) una vista unificada de la información clínica recogida durante el proceso de atención de los pacientes. Esta información no se limita únicamente a la recogida en un departamento sino que incluye la información aportada por la institución o el conjunto de instituciones donde el paciente ha sido atendido alguna vez. Esta situación da lugar al concepto de historia clínica (de salud) electrónica.

Existen tres factores clave que cabe tener en consideración a la hora de afrontar un proyecto de integración de datos, independientemente de que sean clínicos o no. Estos son: autonomía de las fuentes de datos a integrar, su heterogeneidad y su distribución.

Autonomía. Cuando las bases de datos están bajo un control separado e independiente se dice que éstas son autónomas. Aquellos que controlan el acceso a las bases de datos a menudo solo permitirán el acceso a los datos a usuarios ajenos si siguen manteniendo el control total sobre los datos almacenados. Existen distintos tipos de autonomía, todos ellos importantes, y que es necesario entender y saber cómo hacerles frente cuando un sistema de bases de datos participa en una red o bien cuando comparte su información. Los distintos tipos de autonomía son:

– Autonomía de diseño: las bases de datos locales pueden elegir su propio modelo de datos (cómo se estructura la información), lenguaje de interrogación, restricciones, qué operaciones o funciones soporta, etc. Esta es la principal causa de la heterogeneidad entre los distintos sistemas.

– Autonomía de comunicación: las bases de datos tienen el poder de decidir cuándo y cómo responder a las peticiones de información procedentes de otros sistemas.

– Autonomía de ejecución: las bases de datos locales controlan el orden de ejecución de las transacciones u operaciones.

– Autonomía de asociación: las bases de datos locales pueden decidir qué datos y qué funcionalidad comparten determinados usuarios.

Heterogeneidad. La heterogeneidad se debe principalmente al desarrollo independiente de los sistemas de información. La heterogeneidad se presenta en diversos aspectos y por diversas razones. Desde un punto de vista puramente técnico la heterogeneidad está presente cuando existen diferencias en el hardware, sistemas operativos, sistemas de gestión de bases de datos y lenguajes de programación empleados para desarrollar los sistemas de información. Desde el punto de vista conceptual, la heterogeneidad está presente cuando existen diversos modelos de datos y cuando la misma información se interpreta y se modela de forma diferente. Por ejemplo, el uso de un mismo nombre para nombrar diversos conceptos, el uso de dos nombres distintos para referirse al mismo concepto o diferencias en las unidades de medida empleadas.

Distribución. Un tercer problema, ortogonal a los dos anteriores, es el de la distribución física de las fuentes de datos. La distribución aparece cuando las fuentes de datos residen en sitios diferentes. Decimos que un sistema es distribuido si algunos de sus componentes residen en nodos diferentes. En este caso los nodos deben conectarse de alguna forma.

Arquitectura de historia clínica electrónica

Las historias clínicas electrónicas (HCE) contienen información clínica sobre los pacientes. Esta información debe tener alguna estructura, de forma que pueda ser manipulada o procesada por un sistema informático. La estructura debe ser adecuada tanto para el proceso de atención sanitaria como para otros posibles usos (investigación, formación, etc.). Por este motivo uno de los aspectos más importantes a la hora de desarrollar sistemas de historias clínicas es cómo organizar la información clínica.

Una arquitectura de historia clínica electrónica (AHCE) modela las características genéricas aplicables a cualquier anotación en una historia clínica, dicho con otras palabras, es un modelo conceptual de la información que puede estar contenida en cualquier HCE y, por tanto, modela las características comunes a todas las HCE. Por el contrario, no detalla qué información debe estar contenida en una his-

toria clínica ni cómo un sistema de HCE debe implementarse. La arquitectura debe proporcionar principalmente constructores para capturar fielmente el significado original de la información y asegurar que la historia clínica sea comunicable. Se entiende por comunicable que el destinatario de los datos pueda interpretar fielmente el significado original de los datos y procesarlos.

Sin lugar a dudas, **la estandarización de AHCE es esencial si los registros médicos deben ser compartidos o transferidos** más allá de la organización donde fueron creados ya sea para permitir la movilidad del paciente o para facilitar otras acciones como la teleconsulta. De forma muy resumida una AHCE debe cumplir los siguientes requerimientos:

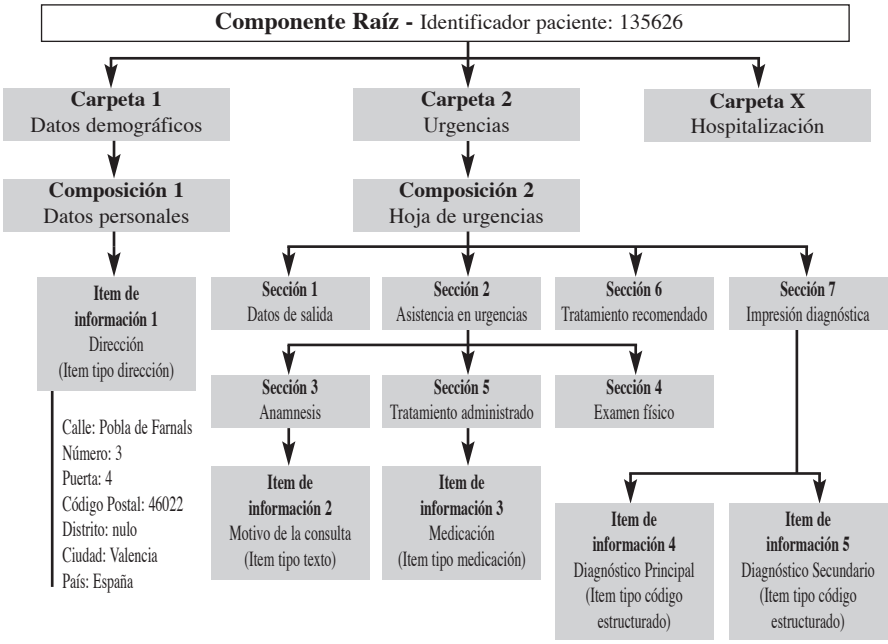
– Capturar fielmente el significado original de una anotación o conjunto de anotaciones en la historia clínica.

– Proporcionar un marco apropiado para analizar e interpretar las historias clínicas.

– Incorporar los constructores esenciales para representar la información médico-legal necesaria para la comunicación fiable de información clínica entre distintos profesionales independientemente de su ubicación.

El comité técnico 251 del Comité Europeo de Normalización (CEN) está realizando en los últimos años un gran esfuerzo para la estandarización de la AHCE. En su 35º Meeting, celebrado en Bruselas en el mes de junio de 1999, aprobó el prestándar ENV13606 que en cuatro documentos establece los principales componentes de la arquitectura. La próxima versión, ya con la categoría de estándar, está prevista para el año 2004. Actualmente, la labor del grupo de trabajo del CEN se centra en hacer converger el estándar lo máximo posible con HL7 y definir un modelo para la definición de arquetipos (definiciones formales de términos del dominio clínico). Cabe destacar que los trabajos de estandarización del comité CEN/TC251 no son un intento de estandarizar el contenido de la historia clínica electrónica, sino de estandarizar el continente en su estructura y los mensajes de intercambio de información clínica, además de proporcionar un mecanismo para el control de la distribución y el acceso. Este estándar se aborda con más detalle en otro capítulo de este informe. La Figura 4 muestra un ejemplo de utilización de una arquitectura de historia clínica para la representación de la historia clínica de un paciente, en concreto este ejemplo está basado en la arquitectura ENV13606 del CEN.

Figura 4. Ejemplo de uso de una arquitectura de historia clínica



Metodologías de integración

La tecnología informática ofrece diferentes soluciones al problema del compartimiento de datos entre aplicaciones. En este apartado se explicará brevemente cuatro de ellas: los basados en mensajes, mediadores, bases de datos federadas y *data warehouse*.

1. Integración basada en mensajes

La comunicación basada en mensajes (por ejemplo basada en HL7, DICOM, o X12N) se suele considerar como una metodología, más que para la integración de datos, para la integración funcional de sistemas de información clínicos tanto dentro como fuera de la organización. Por funcional se entiende que permite la automatización del flujo de información de los procesos médicos, tales como la admisión (por ejemplo, la información demográfica sobre un nuevo paciente es comu-

nicada automáticamente al resto de sistemas), alta o traslado de los pacientes, petición de pruebas diagnósticas o recepción de resultados. Básicamente la comunicación de información clínica y/o administrativa se lleva a cabo de manera muy similar a la tradicional comunicación basada en papel, pero en vez de emplear formularios o cartas se envían mensajes.

La comunicación entre el sistema emisor y el receptor depende completamente del uso de mensajes, es necesario, por tanto, llegar a un acuerdo en cuanto a la sintaxis y semántica. Dicho con otras palabras, la comunicación solo es posible cuando ambos sistemas soportan el mismo estándar. Para transmitir datos en un formato estructurado, es necesario que los sistemas comunicantes alcance un acuerdo en:

– El orden que son transmitidos los ítems individuales

– Qué caracteres separan los ítems, de forma que puedan ser diferenciados

– El formato de los ítems, por ejemplo, cómo se escribe una fecha (sintaxis)

– El significado de los ítems (semántica), por ejemplo, el primer ítem es el nombre, etc.

Desde un punto de vista técnico, la integración basada en mensaje se puede conseguir por medio de un conjunto de servicios *middleware* tales como terminológicos, de seguridad o de nombres, un conjunto de agentes con comportamiento activo (actúan cuando sucede un evento) y un conjunto de componentes software que facilitan la comunicación, como TCP/IP a nivel más básico y HL7 o DICOM a un nivel superior.

2. Mediadores

Los mediadores son programas informáticos especializados que obtienen la información a partir de una o más fuentes de datos o de otros mediadores, es decir, de los componentes que están por debajo de él, y proporcionan información a los componentes que están por encima (otros mediadores) y a los usuarios externos del sistema. Las fuentes de datos están "envueltas" por una capa de software, denominada adaptador o *wrapper*, el cual traduce entre el lenguaje, modelos y conceptos de la fuente de datos y el lenguaje, modelo y conceptos utilizados en el mediador. Un mediador ofrece una vista unificada e integrada de la información que se encuentra almacenada en las diversas fuentes de datos. El mediador no almacena datos, pero pueden ser consultados como si lo hiciese, su tarea es acceder a sus fuentes de datos y encontrar la respuesta a la consulta. Desde un punto de vista teórico un mediador se caracteriza por una arquitectura basada en un esquema global y un conjunto de fuentes de datos. Las fuentes de datos son los repositorios de la

información real, mientras que el esquema global proporciona una vista virtual, integrada y reconciliada de las fuentes de datos subyacentes.

3. Bases de datos federadas

Existe y ha existido un gran interés en la comunidad científica en el desarrollo de sistemas de bases de datos federadas. Una base de datos federada está formada por un conjunto de bases de datos independientes y autónomas, cada una de las cuales tiene sus propios usuarios y transacciones locales y cooperan para formar una federación que permite a los usuarios globales acceder a los datos almacenados en las bases de datos participantes como si estuviesen accediendo a una única base de datos. Cada sistema departamental puede autorizar el acceso a parte de la información que contiene, por medio de un esquema "exportado" el cual describe en un modelo de datos unificado el subconjunto de cada una de las bases de datos que forma parte de la federación. Es posible, también, que las bases de datos participantes en la federación puedan extender su esquema para incorporar subconjuntos de datos pertenecientes a otras bases de datos. Las bases de datos federadas son la solución más completa, escalable y robusta pero a su vez también es la más compleja, es por esto, que en la mayoría de los casos no se desarrollan sistemas con toda la funcionalidad esperada. Por ejemplo, sólo permiten la lectura de datos y no la escritura.

4. Data Warehousing

Un *data warehouse* es un gran repositorio de datos orientado a temas, integrado, no volátil y variante en el tiempo cuyo propósito es el de ayudar en la toma de decisiones. Básicamente un *data warehouse* es una gran base de datos que almacena una **copia** de los datos operacionales de la organización y cuya estructura está optimizada para labores de búsqueda y análisis. El ámbito de los *data warehouse* suele ser toda la organización, por el contrario, si el ámbito es más reducido, por ejemplo, un departamento hospitalario, se suele hablar de *data mart*. Los datos provienen de diferentes fuentes ya existentes en la organización, las cuales pueden ser heterogéneas entre si, y, por tanto, puede tener distintos formatos y tipos. La información extraída se transforma, para eliminar inconsistencias y se almacena en la base de datos del *data warehouse*.

Veamos cada una de las características de un *data warehouse* con más detalles. Decimos que un *data warehouse* está orientado a temas porque la información que contiene está organizada en función de los aspectos que son de interés para la organización, lo cual facilita a los usuarios finales el acceso y compresión de los datos. La integración, es con diferencia, el aspecto más importante de un *data warehouse*. Como

ya se ha comentado las fuentes de datos, como consecuencia principalmente de la autonomía de diseño, son heterogéneas entre sí. La integración es el proceso de eliminar la heterogeneidad presente en el modelado e interpretación de la información. De está forma, cuando los datos se cargan en el *data warehouse* las posibles inconsistencias son eliminadas. Un ejemplo típico es el de la codificación En un sistema el sexo del paciente está representado por las letras "H" y "F", en otro por los números 0 y 1 y en otro por los términos "hombre" y "mujer". No importa la codificación empleada en el *data warehouse*, lo importante es que el sexo se represente de manera consistente independiente de la fuente de datos. Por no volátil se entiende que la información contenida en un *data warehouse* es permanente, solo se realizan dos tipos de operaciones: la carga inicial de datos y el acceso a datos. No hay actualización de datos. Se dice que los *data warehouse* son de tiempo variante porque contiene información histórica que no se puede actualizar. Por tanto, podemos ver la información contenida en un *data warehouse* como una larga sucesión de "vistas instantáneas".

Existen herramientas conocidas como ETL (Extraction, Transformation and Loading), extracción, transformación y carga en castellano. Estas herramientas ayudan en los procesos de extracción de datos de las fuentes de datos, la transformación de los datos para que se adapten a las necesidades de la organización y dan soporte a la inserción de la información en la base de datos del data warehouse. Algunas transformaciones típicas ofrecidas por estas herramientas son: limpieza de datos (por ejemplo, corrección de errores o completar información faltante), conversión (cambios de tipos de datos o en su representación), integración (combinación da datos provenientes de distintas fuentes y eliminación de las inconsistencias que puedan aparecer, transformación a un formato estándar), agregación (generación de datos resumidos según algún criterio, por ejemplo, por meses, departamento, etc.).

Podemos detallar algunos ejemplos de potenciales usos del *data warehousing* en el ámbito de las historias clínicas son:

- Proporcionar acceso a información almacenada en sistemas aislados e incompatibles ya existentes.
- Generación de informes de actividad y mejorar la obtención de estadísticas e información sobre los pacientes.
- Como soporte a la investigación (epidemiología, eficiencia de tratamientos, etc.).

En resumen, un *data warehouse* se diseña con el propósito de almacenar y consultar grandes cantidades de información relacionada. La información proviene de

diversas fuentes, por tanto un *data warehouse* proporciona una vista unificada y consistente de información proveniente de diversas fuentes. Además, proporciona herramientas para la toma de decisiones, ejecución de consultas y generación de informes.

MIGRACIÓN DE SISTEMAS

Introducción

La gran difusión de la informática durante estas últimas décadas ha llevado a muchas organizaciones a una situación en la que actualmente existen numerosos sistemas de información, un poco obsoletos, pero a la vez necesarios en muchas organizaciones de manera que éstos se resisten cada vez más a cualquier modificación o evolución. A este tipo de sistemas se les denomina *legacy*. Suelen presentar numerosos problemas: frágiles, inflexibles, no extensibles, están aislados, no son abiertos, impide ser competitivo (o competente) en el sector, su mantenimiento monopoliza el tiempo y dinero de la organización, etc.

La migración de sistemas es un mundo muy complejo. Un ámbito de investigación relativamente nuevo en el mundo de la informática. Existen numerosas técnicas y guías que describen cuales deberían ser los pasos a seguir en cualquier migración de un S.I. La mayoría de estas guías están basadas en experiencias concretas de empresas. Parece ser que cada migración está estrechamente ligada a la política y al *modus operandi* de cada organización siendo difícil el parametrizar qué pasos exactos se deben dar en una organización, para minimizar al máximo el impacto de la migración en el funcionamiento normal de la organización implicada.

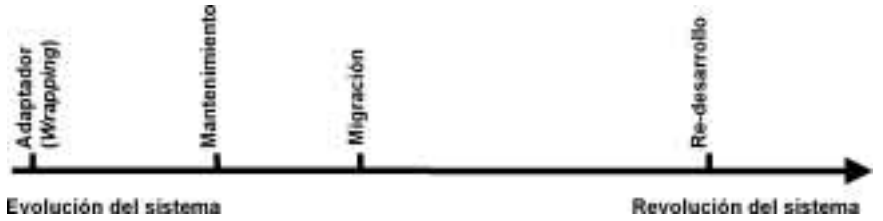
Identificar cuándo es necesaria una migración

En primer lugar hay que identificar cómo se requiere hacer evolucionar el sistema. Gráficamente, Figura 5, se pueden representar las posibles decisiones con un simple dibujo dónde de izquierda a derecha están representadas las posibles intervenciones sobre el sistema, que van desde decisiones moderadas hasta decisiones que pueden causar gran impacto en la organización. A continuación se comentan las diferentes posibilidades.

Adaptador (*wrapping*). Cuando solo se tiene necesidad de ofrecer una entrada o una interfaz para que otros sistemas puedan utilizar la información del *legacy* (sistema a migrar). El impacto que suele tener esto sobre el sistema no suele ser importante porque solo supone crear una capa por encima del sistema sin tocar el mismo. De esta manera los clientes trabajan directamente con este componente o capa (*wrapped component*) que actuará como servidor de las peticiones de los clientes.

Figura 5. Impacto de las intervenciones de migración sobre el sistema

Adaptado de Bisbal J., Lawless D., Wu B., Grimson J. Legacy Information Systems: Issues and directions. IEEE Software, 15 (5), pp. 103-111 (1999).



Mantenimiento. Cuando se requiere hacer un mantenimiento sobre la aplicación para incorporar mejoras o cambios requeridos por los nuevos tiempos y nuevas necesidades el impacto sobre el sistema será mayor, puede llegar incluso a resultar una tarea un tanto osada cuando el código es antiguo y ha sufrido muchos cambios durante su vida útil. Muchos de estos sistemas tienen millones de líneas de código y con el tiempo acaban degradándose hasta el punto que son difícilmente mantenibles.

Migración. Cuando el re-desarrollo es un riesgo inaceptable y poner un adaptador (*wrapper*) no representa una alternativa a tener en cuenta debido a sus limitaciones, la migración del sistema hacia otro entorno más abierto debe abordarse seriamente. Es una tarea difícil pero los beneficios a largo plazo en caso de éxito no son despreciables: mayor flexibilidad, mejor comprensión del sistema, facilidad de mantenimiento, reducción de costos, etc. La esencia de una migración es poder mover un sistema existente y que es operativo a un nuevo sistema intentando mantener como mínimo la misma funcionalidad del *legacy* y además sin causar ninguna interrupción en el sistema que esté operativo en esos momentos.

Redesarrollo. Re-desarrollar el sistema de nuevo es una aproximación que comúnmente se denomina en este campo cómo "*Big bang*" también conocido como *Cold Turkey*. Supone rehacer todo el sistema de información desde cero usando para ello una arquitectura moderna con herramientas y bases de datos ejecutándose en una nueva plataforma de hardware. Esta aproximación requiere mucho esfuerzo en sistemas de gran tamaño. Para la mayoría de las organizaciones el completo re-desarrollo de un sistema no es una opción y se ven forzados a buscar caminos alternativos para hacer evolucionar sus *legacy*.

Ejemplos de estrategias de migración

1. Estrategia "Chicken little"

Esta estrategia de migración permite de manera crítica soportar una migración haciendo interoperar el sistema *legacy* y el sistema nuevo hasta la finalización de la misma. Esto es posible gracias a la utilización de un módulo conocido en general como *Gateway* "un módulo de software introducido entre los componentes software para que interoperen entre ellos". La estrategia de "*Chicken little*" propone un plan con 11 pasos que consisten en lo siguiente:

1. Análisis incremental del sistema de información del *legacy*.
2. Descomposición incremental de la estructura del sistema de información del *legacy*.
3. Diseño incremental de las interfaces a emplear.
4. Diseño incremental de las aplicaciones a emplear.
5. Diseño incremental de las BBDD (bases de datos) a emplear.
6. Instalación del entorno del sistema objetivo.
7. Creación e instalación incremental de los "*Gateways*" necesarios.
8. Migración incremental de las BBDD *legacy*.
9. Migración incremental de las aplicaciones *legacy*.
10. Migración incremental de los interfaces del *legacy*.
11. Uso y paso incremental hacia el sistema de información migrado.

Con el uso de esta estrategia progresivamente se va migrando el sistema utilizando para ello herramientas y tecnología moderna. El sistema migrado inicialmente será muy pequeño pero poco a poco irá creciendo según vaya progresando la migración. En el momento en el que el sistema migrado pueda realizar por sí solo las mismas operaciones del *legacy* se da por concluida la migración pudiendo prescindir del *legacy*. Mientras esto no ocurre el *legacy* y el sistema en migración conviven e interoperan a través del uso de los *Gateways*.

2. Metodología de migración Butterfly

Este método descarta el uso de *Gateways* y asume que durante el proceso de migración el *legacy* va a estar totalmente operativo durante todo el tiempo y no va a necesitar interoperar con el sistema migrado.

Hace especial hincapié en la migración de la información contenida en las BBDD del *legacy*. El desarrollo del sistema a migrar es un aspecto totalmente separado de la migración de la información. Cuando la migración de datos se ha completado entonces se bloquea la información del *legacy* a solo lectura. A partir de ese momento la manipulación de la información se realiza sobre las BBDD migradas utilizando para ello una serie de depósitos de almacenamiento temporal de la información. A continuación, y normalmente en paralelo, se construye el sistema migrado y se traspasa la información del *legacy* junto con la de los depósitos a las nuevas BBDD. Siempre queda un depósito que no se migra, en el cual el sistema, que siempre se ha mantenido operativo, escribe. Llegado el momento en el que solo queda un depósito entonces se puede parar el sistema durante unos minutos y completar el traspaso total de la información.

Con esta metodología el *legacy* solo permanece apagado durante un insignificante lapso de tiempo y se evita la costosa tarea de crear *Gateways* con el propósito de interoperar para evitar así este paro.

Justificación para una migración

Siempre debe de ser visto como el último recurso. Por tanto, antes de tomar cualquier decisión es conveniente hacer un estudio intensivo de la situación para evaluar los riesgos y los beneficios de una migración

Por tanto, hay que entender a fondo el *legacy* antes de realizar cualquier intento de migración ya que su conocimiento es fundamental para especificar los requerimientos para el nuevo sistema a migrar. Quizás uno de los principales impedimentos para esta fase sea el hecho de que habitualmente los *legacy* están pobremente documentados por no decir que en muchas ocasiones la documentación es nula.

Hay que migrar las estructuras de datos y, por tanto, también es necesario su entendimiento. Una vez entendidas estas estructuras, se deben identificar las redundancias de datos para migrar solamente la información necesaria.

Hay que elegir la estrategia que mejor se adapte a la organización, es interesante buscar en la literatura experiencias similares para no caer en los mismos errores.

Migración al nuevo sistema de información

El nuevo sistema de información, cómo mínimo, debe de ser equivalente en funcionalidad al *legacy* pero esta vez en un sistema abierto. Sería interesante que, además, fuera multiplataforma y que utilizara tecnologías interoperables con la

mayoría de sistemas. Una de las primeras decisiones importantes será escoger la arquitectura a la que debemos migrar. Debería ser prioritario escoger una que permita un mantenimiento y escalabilidad en el futuro, para que el sistema no se convierta en un futuro muy próximo en otro *legacy*. Además, dada la importancia y el crecimiento de la actual WWW, el sistema objetivo debe facilitar o estar plenamente integrado en ella.

Testeo del sistema migrado

Cerca del 80% del tiempo de la migración se dedica al testeo. No es recomendable incorporar nuevas funcionalidades al sistema migrado hasta no probar a fondo que éste responde con todas las funcionalidades más importantes. Aunque uno de los atractivos de haber aceptado el riesgo de la migración es la promesa de poder incorporar nuevas funcionalidades es mejor dejar esto para el final para no entorpecer comparaciones directas y de manera paralela entre el *legacy* y el sistema migrado que serán de gran ayuda.

Fase de migración

Comprende la fase en el que se abandona el uso del legacy por el nuevo sistema. Es una misión crítica cuyo proceso puede ocasionar un paro en el sistema.

Finalmente hay diferentes estrategias de transición para culminar la migración.

1. *Cut-and-Run*. Apagar el *legacy* y enchufar el nuevo sistema. En muchos casos esto no es una opción real o posible.
2. Incremental y progresiva. En cada paso se reemplazan unos pocos componentes del *legacy* hasta que se completan todos los pasos.
3. Estrategia de operaciones paralelas. El *legacy* y el sistema objetivo operan simultáneamente con ambos sistemas ejecutando todas las tareas posibles. Cuando el sistema migrado es totalmente fiable, entonces se puede desechar el *legacy*.

El campo de la migración de *legacy's* es un tema relativamente nuevo en la comunidad investigadora, aún se requieren muchos esfuerzos y estudio en todos los aspectos relacionados con la migración. Se necesitan esfuerzos en identificar un mayor número de tipos de *legacy* y desarrollar procesos y metodologías para cada uno de ellos.

También hacen falta un mayor número de herramientas que faciliten y ayuden en todos los procesos necesarios para llevar una migración con éxito.

XML

Introducción

XML son las siglas de *eXtended Markup Language* o lenguaje extendido de marcado, en castellano. Básicamente es un lenguaje para describir información, no es un programa software y por tanto no realiza ninguna tarea por sí mismo. Como su propio nombre indica, es un lenguaje de marcado. Un lenguaje de marcado es un método para describir un documento insertando etiquetas en él. Las etiquetas nos dan información sobre el contenido del documento. Por ejemplo, HTML es otro lenguaje que posee un conjunto de etiquetas que indican al navegador (Netscape, Internet Explorer, Mosaic, etc) cómo presentar la información. HTML posee etiquetas, por ejemplo, para indicar que el texto debe presentarse en negrita, cursiva, en un tamaño mayor o menor según la relevancia que se desee. HTML se ha mejorado mucho desde su creación, sin embargo, su principal desventaja es que el conjunto de etiquetas es fijo. XML soluciona este problema de flexibilidad.

XML es un lenguaje de marcado extensible, lo que significa que con XML podemos definirnos nuestras propias etiquetas personales según nuestras necesidades. Realmente XML no es un lenguaje de marcado, sino un lenguaje para crear nuestro propio lenguaje de marcado.

El *abuelo* de XML es SGML (Standard Generalizad Markup Lenguaje), el cual es una norma ISO desde 1986. SGML es muy potente, pero a su vez es muy complejo. Técnicamente XML es un descendiente directo de SGML, pero mucho más simple. XML está libre de toda restricción de propiedad intelectual, no existen patentes, marcas registradas o copyright, esto es así porque es una especificación creada por el W3C (World Wide Web Consortium) al igual que HTML. W3C define muchos de los estándares relativos a la web. A diferencia de ANSI o ISO no genera normas oficiales. Por está razón el consorcio emite su opinión en forma de recomendaciones y no como estándares internacionales. Sin embargo, las recomendaciones de este organismo suelen acabar en lo que se denomina estándares de facto. El desarrollo de XML comenzó en 1996 y desde febrero de 1998 es una recomendación del W3C. XML nació con la idea de combinar en un lenguaje la potencia de SGML y la simplicidad de HMTL. Para estar al día de los trabajos llevados a cabo por el W3C en relación a XML se puede consultar la página *http://www.w3.org/XML*.

Estructura básica de un documento XML

Supongamos que se quiere representar información demográfica sobre los pacientes, en particular, se tiene un paciente con número de historia clínica 234561 y de nombre Ramón García Benlloch que vive en la Calle Cádiz, número 14 de Valencia. Supongamos, también, que tenemos la necesidad de intercambiar esta información con otro centro. Un documento XML que representa esta información y que puede ser enviado al destinatario tendría un aspecto similar al del ejemplo 1.

Ejemplo 1

```
<?xml version=”1.0” ?>
<paciente>
        <nhc> 234561</nhc>
        <nombre_completo>
                <nombre> Ramón </nombre>
                <apellido1> García ></apellido1>
                <apellido2> Benlloch </apellido2>
        <nombre_completo>
        <direccion tipo=”domicilio”>
                <calle> Cádiz </calle>
                <numero> 14 </numero>
                <poblacion> Valencia </poblacion>
                <provincia codigo=”46”> Valencia </provincia>
        </direccion>
</paciente>
```

Como puede comprobarse en el ejemplo 1, un documento XML puede ser entendido fácilmente por un humano, aunque su propósito último es que sea procesado por un ordenador.

Veamos a continuación como interpretar este documento. Un documento XML se puede dividir en dos bloques: el **encabezamiento**, que proporciona al destinatario del documento (generalmente una aplicación informática) información sobre cómo manejar el documento y el **contenido** que son los datos representados en el documento.

El encabezamiento es simplemente una declaración XML de la forma: <?xml version=”1.0” ?>. El cual indica la versión de XML utilizada. También puede incluir la codificación de caracteres empleada o si el documento necesita de otros para ser entendido en su totalidad:

<?xml version=”1.0” encoding=”UTF0” standalone=”no” ?>.

En XML las etiquetas van encerradas entre los símbolos ‘<’ y ‘>’. Existen dos tipos de etiquetas la de apertura y las de cierre. Las de cierre se distinguen de las de apertura en que tras el símbolo ‘<’ aparece el símbolo ‘/’. Así por ejemplo, la etiqueta <calle> es de apertura y </calle> es de cierre. Toda etiqueta de apertura tiene que tener una etiqueta de cierre. Es fácil de deducir que el extracto del documento “<calle> Cádiz </calle>” nos dice que Cádiz es el nombre de una calle. La gran potencia de XML reside en el hecho en que los usuarios pueden definirse sus propias etiquetas.

Los elementos, también llamados nodos, son las estructuras primarias de un documento XML. Los elementos más básicos que contienen información están compuestos por una etiqueta de apertura, un texto y una etiqueta de cierre. Ejemplos de elemento son: <calle> Cádiz </calle> o <apellido2> Benlloch </apellido2>. Los elementos pueden contener a otros elementos, formando jerarquías, lo cual lo hace muy útil para representar información que tiene una estructura jerárquica, como las historias clínicas. El siguiente extracto del documento XML del ejemplo 1, representa un elemento con etiqueta <nombre_completo>:

```
<nombre_completo>
      <nombre> Ramón </nombre>
      <apellido1> García ></apellido1>
      <apellido2> Benlloch </apellido2>
<nombre_completo>
```

Además del texto, contenido entre las etiquetas un elemento también puede tener atributos, los cuales se declaran en la etiqueta de apertura del elemento. Los atributos proporcionan información adicional sobre el elemento. Un atributo tiene dos partes: el nombre y el valor. Por ejemplo, en <provincia codigo=”46”>, el nombre es *codigo* y el valor *46*. Puede parecer que los atributos pueden representarse como elementos de texto, y de hecho es así. No hay reglas claras para determinar qué modo de representación es el más adecuado.

Tecnologías asociadas

Durante estos años desde la aparición de XML, el W3C ha desarrollado un amplio abanico de tecnologías asociadas a XML. Este apartado tiene como propósito dar una pequeña descripción de alguna de ellas.

DTD (Document Type Definition). Es un lenguaje que sirve para dos propósitos: proporcionar una sintaxis para definir la estructura de un documento XML y

validar un documento cuando se asocia un DTD. Un documento XML es válido si sigue la estructura y restricciones definidas en el DTD.

XML Schema. Cumple el mismo propósito que los DTD, con la diferencia de que un XML Schema es a su vez un documento XML y es mucho más potente que los DTD.

XPATH. Como hemos visto los documentos XML son estructuras jerárquicas que pueden ser muy complejas. XPATH es un lenguaje que nos permite "navegar" dentro de los documentos para alcanzar aquellos elementos que nos sean de interés. XPATH se utiliza en otros estándares XML.

XSLT (Extensible Stylesheet Language Transformation). Define un lenguaje para transformar un documento XML en otra representación textual diferente, generalmente otro documento XML, aunque existen otras transformaciones posibles, como por ejemplo a HTML para publicación en Web e incluso RTF o PDF.

Las siguientes dos tecnologías están relacionadas con la programación de aplicaciones que usa XML.

DOM (Document Object Model). DOM es una API (interface de programación de aplicaciones) para la navegación por documentos XML. En DOM un documento se ve como un árbol de nodos, para lo cual el documento se *parsea* (se comprueba que está bien construido) y se almacena en la memoria del ordenador.

SAX (Simple API for XML). Al igual que DOM, es una API que permite *parsear* un documento XML. La diferencia estriba en que SAX no crea ninguna estructura de datos sino que se basa en eventos, cada vez que se encuentra un elemento, SAX lanza un evento que debe ser interpretado por la aplicación y ésta, en función del tipo de evento, realiza la acción pertinente. En general es más eficiente que DOM, y está especialmente indicado para documentos muy grandes.

Principales áreas de aplicación

XML se está empleado en numerosos ámbitos, desde el comercio electrónico hasta la gestión del conocimiento. Podemos nombrar tres usos de especial relevancia para la gestión de historias clínicas electrónicas: documentación, bases de datos y el más importante de todos, el intercambio de información entre aplicaciones. Veamos con algo de detalle cada uno de ellos.

Documentación

En un principio XML se consideraba fundamentalmente un lenguaje para describir metacontenidos, es decir información relativa al contenido de un documento, como título, autor, tamaño, fecha de creación, palabras clave, etc. Actualmente, su uso va más allá, y se utiliza para la gestión de conocimiento (redes semánticas, ontologías, tesauros), procesamiento del lenguaje (diccionarios) o edición y publicación electrónica.

Bases de datos

El uso de XML como modelo de datos para el desarrollo de sistemas de bases de datos es un área en continuo desarrollo. El objetivo que se persigue es almacenar, gestionar y consultar de forma eficiente documentos XML. Otro uso de XML es el de herramienta para la integración de datos. Podemos representar los modelos de datos de los sistemas a integrar en XML. Así por ejemplo, un DTD puede ser utilizado como descriptor de las estructuras de datos utilizadas por las fuentes de datos.

Intercambio de información

La promesa de XML consiste en proporcionar un medio sencillo para que aplicaciones distintas, ya estén dentro o fuera de la organización emisora, puedan comunicarse. Cabe destacar que el simple uso de XML no permite la interoperabilidad entre las aplicaciones, para conseguirla es necesario, además, que la aplicación receptora pueda interpretar adecuadamente el documento XML recibido, para lo cual es necesario que el receptor y el emisor lleguen a un acuerdo en cuanto a la estructura y etiquetado de los documentos XML.

XML ofrece diversas ventajas a la hora de intercambiar extractos de historias clínicas electrónicas entre diversas aplicaciones. En otras podemos citar:

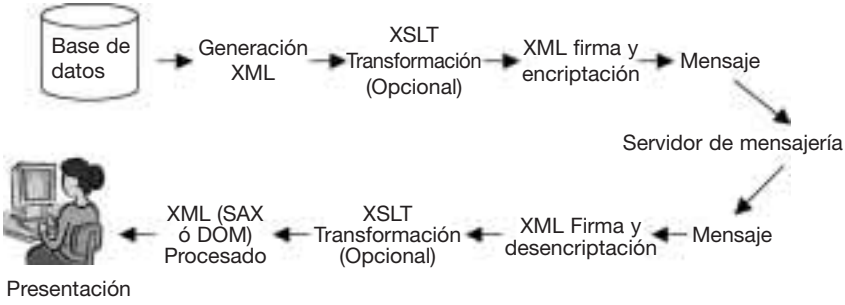
- XML es independiente del software y hardware empleado.
- Las historias clínicas suelen tener una estructura jerárquica al igual que XML lo que lo hace muy adecuado para representar extractos de historias clínicas.
- Las tecnologías asociadas como DTDs y XMLSchema permiten definir modelos públicos de mensajes para el intercambio de información. Es decir, con un DTD público, se puede describir cómo son los mensajes (etiquetas y estructura) y facilitar la compresión y procesado a la aplicación destinataria. Así por ejemplo, en el prestándar europeo para la comunicación de la historia clínica electrónica en su parte IV, donde se especifican los mensajes que per-

miten el intercambio de historias clínicas electrónicas entre centros o personal sanitario, se adjunta un borrador de DTD que describe cómo deben ser los mensajes para que sean compatibles con la norma.

– La separación de contenido (documento XML) y reglas de presentación (XSLT) permite una gran flexibilidad a la hora de desarrollar sistemas de comunicación de historias clínicas. Por ejemplo, la presentación de la información puede ser responsabilidad del receptor, del emisor o de ambos.

La Figura 6 muestra un escenario típico del uso de XML para el intercambio de información entre aplicaciones sanitarias. Los datos están almacenados en una base de datos remota, los datos requeridos se extraen y se genera un documento XML. Este documento puede ser transformado por medio de XSLT y a continuación se firma y se encripta. Algún servidor de mensajería se encarga de construir un mensaje, por ejemplo SOAP, que contiene al documento XML y lo envía a la aplicación destino, la cual realizará un desencriptado, una transformación, si es necesaria, lo analizará y lo presentará al usuario final o lo almacenará en una base de datos.

Figura 6. Ejemplo típico del uso de XML para el intercambio de información



ONTOLOGÍAS

En informática las ontologías tienen su origen en la rama de la inteligencia artificial donde se utilizan para facilitar la representación, compartición y reutilización de conocimiento. Desde la última década del siglo pasado las ontologías son uno de los campos de investigación más activos. Recientemente, no sólo se circunscriben al ámbito de la inteligencia artificial, sino cada vez son más empleado en otros campos como la integración inteligente de información, sistemas de información cooperativos, extracción de información, comercio electrónico y gestión del cono-

cimiento. Esto es debido principalmente a la promesa que ofrecen: un entendimiento compartido de algún domino que puede ser comunicado entre personas y máquinas. Esto es aún más importante en la situación actual donde los ordenadores ya no son máquinas aisladas sino que cada vez más son puntos de entrada a una red mundial de intercambio de información, tendencia a la que no es ajeno el sector sanitario. Por tanto, es esencial disponer de herramientas que faciliten este intercambio de información y conocimiento.

Existen múltiples definiciones de ontología, una de las más extendidas y citadas se debe a Tom Gruber: Una ontología es una especificación explícita de una conceptualización. Basada en esta definición muchas otras se han propuesto. Dieter Fensel propone la siguiente: Una ontología es una especificación formal y explícita de una conceptualización compartida. Podemos definir una conceptualización como un modelo abstracto de un fenómeno del mundo real el cual identifica los conceptos relevantes de ese fenómeno. En este contexto, "formal" se refiere a la necesidad de que las ontologías sean compresibles por las máquinas y no únicamente por humanos. "Explícita" significa que los tipos de conceptos usados y las restricciones sobre su uso se definen explícitamente. Finalmente, "compartida" se refiere al tipo de conocimiento contenido en las ontologías, esto es, conocimiento consensuado y no privado. Otra definición más intuitiva y compatible con la definición anterior podría ser: una ontología es un conjunto de términos de interés en un dominio de información y las relaciones que existen entre ellos. Las ontologías, por su naturaleza y propósito, deben ser creadas y mantenidas por expertos en el dominio.

Como ya se ha comentado el dominio de la información sanitaria es muy complejo donde existen multitud de términos y la estructura de la información puede ser muy compleja. No es de extrañar que haya un gran interés por el desarrollo de ontologías en el ámbito médico. En este sentido, la principal problemática es conseguir una comunicación no ambigua de conceptos médicos detallados y complejos. Por ejemplo, las ontologías médicas se introducen para resolver problemas como la petición, compartición y reutilización de datos de los pacientes, su transmisión y la necesidad de criterios basados en semántica para realizar estudios estadísticos. Dos de los ejemplos más representativos son GALEN y UMLS. GALEN (Rector et al, 1995) incluye un modelo semánticamente válido de terminología clínica representado en un lenguaje formal. El Unified Medical Language System, conocido por las siglas UMLS (http://www.nlm.nih.gov/research/umls/) es una base de datos diseñada para unificar terminologías biomédicas de fuentes dispares.

Una de las grandes ventajas de los ordenadores respecto al papel es su capacidad de procesamiento automático de la información. Actualmente la explotación de esta capacidad de procesamiento se utiliza en el campo de las historias clínicas para cosas tan diversas como la gestión de indicadores, alertas, extracción de información, cálculos estadísticos, sistemas de ayuda a la decisión, sistemas expertos etc. Mirando hacia el futuro, aún podemos dar un paso más allá si dotamos a nuestros sistemas de historias clínicas de la capa semántica necesaria para poder extraer conocimiento y no solamente información. Con esta capa sobre los sistemas facilitaremos en gran medida la interoperabilidad, integración y reutilización de la información, asegurando el significado, compresión y calidad de la información. Las ontologías pueden ser la herramienta para la especificación de esta capa semántica.

Cada vez más aparecen nuevos lenguajes para la especificación de ontologías. Los lenguajes ontológicos son los vehículos para expresar ontologías de forma comprensible por las máquinas, puede encontrarse un estudio sobre estos lenguajes en (Gómez-Pérez y Corcho 2002) y (Fensel 2004). Algunos de los lenguajes más populares son RDF (Resource Description Framework), XOL (XML-based Ontology Exchange Language) desarrollado por la comunidad bioinformática de EEUU, OIL (Ontology Interchange Language), DAML+OIL y últimamente la recomendación de la W3C: OWL (Web Ontology Language). OWL tiene más capacidad expresiva que los otros lenguajes y está diseñado para aplicaciones que además de representar el conocimiento necesiten procesarlo, los lectores interesados en OWL puede consultar el portal de la W3C: http://www.w3.org/2001/sw/WebOnt.

También existen cada vez más herramientas ontológicas que nos permite sacar el máximo partido de las ontologías. Concretamente, existen herramientas para:

– Editar y construir de forma semi-automática ontologías como Protége de la Universidad de Stanford (http://protege.stanford.edu/index.html), OntoEdit de la Universidad de Karlruhe y comercializado por Ontoprise y WebOnto (http://kmi.open.ac.uk/projects/webonto/).

– Reutilizar y combinar ontologías, es decir, creación de nuevas ontologías a partir de las ya existentes. Un Ejemplo de este tipo de herramienta es Chimaera (McGuinness et al. 2000).

MIDDLEWARE

El término "*Middleware*" se utiliza para describir una amplia variedad de software, es por esto, que es difícil dar una definición exacta y concreta de este término. Si echamos un vistazo a la literatura informática nos encontraremos con múlti-

ples definiciones, que van desde una capa software entre el sistema operativo (y/o red) y las aplicaciones hasta el "pegamento" capaz de conectar dos aplicaciones. Una posible definición genérica podría ser aquel software que permite que múltiples procesos (aplicaciones) que se ejecutan en una o en varias máquinas interactúen o se comuniquen entre sí. Este software, por tanto, facilita la comunicación en un sistema distribuido.

Podemos dividir los servicios middleware en tres categorías principales: servicios de integración, servicios de aplicación y servicios de gestión. Los primeros permiten la comunicación entre aplicaciones. Los segundos facilitan el acceso a los servicios distribuidos y a la red, uno de los más importantes es el middleware de acceso a datos, el cual permite a las aplicaciones comunicarse y acceder a bases de datos ya sean éstas locales o remotas. Por último, los terceros gestionan e integran múltiples tipos de middleware con servicios de valor añadido, como la recuperación de errores del sistema, optimización de recursos o el tratamiento de excepciones.

Es este apartado nos centraremos en los servicios middleware para la integración de aplicaciones las cuales ofrecen interfaces a una gran variedad de aplicaciones. Dentro de esta categoría se encuandran: RPC (Remote Procedure Calls) o llamadas a procedimiento remoto, distributed Computing Enviroment (DCE) de la Open Software Foundation (OSF) basada en RPCs, el modelo de componentes de objetos (COM, Component Object Model) de Microsoft, las JavaBeans y las Enterprise Java Beans (EJB) de Sun Microsystems, RMI (Remote Method Invocation) o invocación remota de métodos, CORBA con su especialización para el entorno sanitario CORBAmed y los Web Services. Veamos con mayor detalle los dos últimos: CORBA y Web Services.

CORBA

El Object Management Group (OMG) es una organización sin ánimo de lucro que tiene como misión el desarrollo de estándares para la integración de sistemas por medio de la tecnología orientada a objetos. OMG es el mayor consorcio de software que existe en el mundo y está formado por más de 800 compañías. Uno de los mayores logros de esta organización ha sido el desarrollo del estándar CORBA (Common Object Request Broker Architecture).

CORBA define la infraestructura para la arquitectura OMA (Object Management Arquitecture) de OMG (Object Management Group), especificando los estándares necesarios para la invocación de métodos sobre objetos en entornos distribuidos y heterogéneos. De manera intuitiva podemos ver a un objeto como

una "pieza de software" que proporciona uno o varios servicios y a un método como uno de los servicios que ofrece el objeto.

CORBA ha logrado parte de su éxito a la clara separación entre la interface el objeto, la cual define qué servicios ofrece el objeto y cómo invocarlos y la implementación. Las interface se definen por medio de un lenguaje propio conocido como IDL, el cual posee un alto nivel de abstracción, lo que le hace independiente del entorno de desarrollo, por tanto, para la implementación de los objetos se puede utilizar cualquier lenguaje de programación siempre y cuando éste proporcione enlaces con el lenguaje IDL.

CORBA se basa en el concepto de ORB (Object Request Broker), el cual proporciona mecanismos transparentes para la comunicación entre objetos. CORBA especifica la interfaz que ha de proporcionar todo ORB. Por medio del ORB un objeto cliente puede invocar un método de un servidor objeto, el cual puede residir en cualquier máquina de la red. El ORB intercepta la petición y se responsabiliza de encontrar un objeto que pueda servir la petición, pasar los parámetros de la llamada, invocar el método y devolver el resultado. El cliente no tiene que preocuparse de la localización del objeto invocado, del lenguaje de programación empleado para su implementación o del sistema operativo. Los roles de cliente/servidor se utilizan únicamente para la coordinación de las interacciones entre dos objetos, de hecho, los objetos comunicados a través de un ORB puede ser tanto clientes como servidores dependiendo de la ocasión.

Como paso posterior al desarrollo de CORBA, OMG ha puesto en marcha una serie de grupos de trabajo con el propósito de adaptar este estándar a un conjunto de sectores entre los cuales se encuentra el sanitario. CORBAmed es el grupo de trabajo que OMG ha creado para la adaptación de CORBA al sector sanitario. CORBAmed es una división de CORBA encargada de definir una serie de servicios o middleware básico en el dominio de la sanidad. La misión de este grupo de trabajo se resume en estos dos puntos:

– Mejorar la calidad de la atención sanitaria y reducción de costes por medio del uso de tecnologías CORBA.

– Definición de interfaces estandarizadas orientadas a objetos entre servicios y funciones sanitarias.

Para el ámbito sanitario dónde habitualmente conviven un sinfín de sistemas distribuidos y heterogéneos CORBAmed intenta dar una especificación de los servicios necesarios para que estos sean interoperables entre sí. Siete de los servicios más importantes son:

1.- Servicio de Identificación de paciente. Permite la asociación unívoca de diferentes agregados de información clínica a un único paciente. Se mantiene un índice maestro de paciente encargado de establecer las diferentes correlaciones de Identificación de pacientes involucrados en los diferentes sistemas de información. Esta es una tarea muy importante y crítica siendo la principal barrera en el desarrollo de entornos fiables para la consecución de la HCS.

2.- Servicio de terminología léxico-médica. Usado para:

– Adquisición de la información: asistencia en el proceso de introducción de información codificada.

– Mediación: Transformación de mensajes o elementos de información desde una forma de representación a otra.

– Indexación e inferencia: averiguar si algunas asociaciones pueden existir entre varios elementos de información.

– Manipulación de conceptos compuestos: Ayudar en la entrada, validación, traducción y simplificación de conceptos compuestos.

3.- Servicio de acceso a información multimedia. Usado para extraer y gestionar recursos multimedia entre ellos por ejemplo imágenes médicas. Permite la recuperación y extracción entre sistemas heterogéneos que quieran interoperar entre sí.

4.- Servicio de decisión de acceso a los recursos. Usado para obtener autorizaciones, permisos de administración y acceso sobre la información médica.

5.- Servicio acceso a observaciones clínicas. Implementa la interfaz estandarizada de acceso público a la información clínica de una federación de sistemas heterogéneos. Requiere la implementación de pasarelas estandarizadas para cada sistema de información clínico conectado, para importar, exportar, propagar, indexar los registros clínicos del paciente.

6.- Servicio de gestión de la información abreviada. Usado para compilar y gestionar resúmenes médicos para poder transmitirlos entre sistemas dispares. Este servicio es opcional.

7.- Servicio para facilitar la interpretación de los datos. Usado para la ayuda en la toma de decisiones. A partir de la información de salud extraída de las diferentes fuentes. Este servicio es opcional.

Web Services

Un *Web Service* (WS) es una aplicación que puede ser descrita, publicada, localizada e invocada a través de una red, generalmente Internet y utilizando una mensajería estándar basada en XML.

El protocolo encargado de gestionar esta mensajería se denomina SOAP (*Simple Object Acces Protocol*). Este protocolo define una estructura estándar, reglas de codificado y asociación para transportar documentos XML a través de otra serie de protocolos cómo HTTP, SMTP, FTP. Generalmente se usa SOAP y los WS sobre HTTP.

SOAP permite la comunicación entre aplicaciones heterogéneas de modo que clientes de diferentes plataformas o lenguajes de programación puedan interoperar entre sí.

Los WSs (*Web Services*) son un mecanismo para construir aplicaciones distribuidas (*sin estado, stateless*) que puedan compartir e intercambiar información entre ellas utilizando como protocolo de comunicación SOAP. Paralelamente, alrededor de los WSs existen una serie de protocolos y mecanismos adicionales para facilitar tareas tales como el descubrimiento de WSs distribuidos a lo largo de la Red, o UDDI (*Universal Description, Discovery and Integration of Web Services*), así como una descripción de su uso WSDL (*Web Services Description Languages*) que sirve de ayuda a los programadores para conocer la manera adecuada de hacer uso del WS.

Es un error pensar que los WS constituyen la herramienta ideal para el diseño e implementación de cualquier sistema de información distribuido, de hecho, son herramientas similares a otras tecnologías asentadas hace años, tipo CORBA, RMI, DCOM etc., pero en la actualidad parecen ser el punto de referencia en estándares de comunicación de aplicaciones informáticas.

De hecho, la creación de los WSs no proviene del campo de los sistemas de objetos distribuidos (CORBA). Entre las diferencias más notables que podemos encontrar tenemos los siguientes aspectos:

– Realmente, cuando se hace uso de un WS desde una aplicación cliente no se tiene ninguna referencia a objeto remoto alguno. Simplemente, es un intercambio de mensajes XML en una petición-respuesta clásica. En cambio, mediante tecnologías verdaderamente de objetos distribuidos se posee una referencia al objeto remoto y siempre se puede saber el estado de dicho obje-

to durante su vida útil, incluso pasar esta referencia a otro programa, es lo que en computación distribuida se denomina sistema *stateful* o con estado.

– Otra diferencia significativa es la lentitud de los WSs, al hacer uso de SOAP, que básicamente es un mecanismo de *marshalling* (conversión a texto) para llamadas a procedimientos remotos o RPC (*Remote procedure call*) basado en XML (ASCII). Este marshalling casi siempre resulta más extenso que otros mecanismos, también binarios, como CDR o XDR, lo cual supone que se tenga que enviar más datos por el canal de comunicación. Además, al ser SOAP un protocolo ASCII, los datos necesitan ser convertidos a cadena de caracteres para ser transmitidos en su forma binaria, con el consiguiente consumo de ciclos de procesador.

– En el ámbito de las aplicaciones distribuidas *stateless* (sin necesidad de mantener estado) es dónde se empieza a vislumbrar el potencial de los WSs, dado que los tiempos y complejidad de desarrollo son menores a los de otras tecnologías de objetos distribuidos. Además, si se tiene en cuenta el hecho de que en los entornos hospitalarios suele haber bastantes medidas de seguridad contando en sus infraestructuras de red con *firewalls* y servidores que tienen cortados numerosos puertos de comunicación, los WSs se adaptan perfectamente a esta situación ya que operaran normalmente sobre protocolos como HTTP, SMTP...y difícilmente se pueden encontrar cerrados estos puertos ya que se utilizan a su vez para otros servicios básicos como son la Web y *e-mail* respectivamente.

En forma esquemática las ventajas y desventajas de los WSs son:

Ventajas de los WSs

– El protocolo SOAP bajo el cual se implementan los WSs corre bajo tecnologías abiertas no dominadas por ningún fabricante de tecnología en particular, por lo menos de momento. Lo que no sucede en tecnologías de objetos distribuidos tales como RMI asociadas a la arquitectura Java y la empresa SUN o DCOM asociada a Microsoft.

– Pueden trabajar en cualquier entorno que se permita el acceso de HTTP puerto 80. O sea, la mayoría de sitios. Sin necesidad de reconfigurar firewalls para activar puertos específicos.

– Se consiguen aplicaciones distribuidas bajamente acopladas.

– Desarrollo rápido de aplicaciones en comparación con otras tecnologías distribuidas.

Desventajas

– No *stateless* (sin estado), no interacciones *stateful* (con estado), no posee instancias transitorias (Factorías), gestión del ciclo de vida y notificación de cambios de estado.

– Más lento que otras tecnologías

– Tecnología con muchos estándares satélites aún no asentados o especificados totalmente

INTERFAZ, ERGONOMÍA Y USABILIDAD

Introducción

Cada fabricante especifica sus propias guías de diseño de interfaces informáticas. No obstante, la mayoría de ellos coinciden en algunos principios que parecen ser básicos y bastante bien aceptados en el mundo del diseño de interfaces informáticas.

Si la interacción de un sistema con un usuario no es fácil y confortable, como consecuencia puede ser que el rendimiento del sistema en su globalidad no sea satisfactorio. Ajustar y afinar este sistema una vez finalizado puede ser más costoso que haber invertido desde el principio el tiempo necesario en estos aspectos. Según los expertos en esta materia al menos el 10% del coste de un proyecto debería ser invertido en usabilidad, ergonomía y diseño.

Un buen diseño de interfaz de usuario es aquel construido bajo principios y procesos de desarrollo que se centran en los usuarios y sus tareas.

Se debe rechazar la idea de que cuanto mayor es la complejidad de un sistema más difícil es que se pueda tener una interfaz simple. El sistema siempre va a contener partes simples que pueden ser descompuestas en tareas sencillas.

La interfaz no es solo aquello relacionado con ventanas, botones y eventos de ratón. Es un concepto un tanto más rico, implica todos los mecanismos mediante los cuales podemos llevar a cabo tareas en un sistema. De este modo el poder interactuar por reconocimiento de voz con un sistema también es parte de la interfaz, también lo es la posibilidad de ejecutar una misma tarea de varias formas diferentes, etc.

Recomendaciones generales

Como recomendaciones de diseño de una buena interfaz la mayoría de las guías de diseño suelen coincidir en estos aspectos.

– Mantener simple lo que es simple.

– Hacer el diseño centrado en el usuario (*human-centered y user-centered*).

– Asegurarse de que el diseño de la interfaz está de acuerdo con acciones y hechos que el usuario realizará habitualmente en la aplicación.

Algunos expertos creen que las herramientas actuales no facilitan la innovación en los diseños. Están motivadas más por el desarrollo rápido y automatizado de aplicaciones que por una preocupación real de crear una buena interfaz. De hecho, incluso algunas recomendaciones de los propios fabricantes son demostrablemente incorrectas pero las hacen por la necesidad de compatibilidad con versiones y diseños anteriores.

De este modo queda en manos del usuario el sopesar si es bueno el uso de paradigmas de familiaridad (recomendaciones de fabricantes) en pro de la productividad o por el contrario abandonar desde el principio éstas con la consecuencia de un tiempo de desarrollo mayor en las aplicaciones pero apostando por una mejora sustancial en la usabilidad del producto.

Ante este tipo de decisión si los usuarios emplean la mayoría del tiempo en operaciones rutinarias del producto, dónde aprender a realizar éstas es solo una pequeña porción del escenario, el diseño orientado a la productividad es a menudo la decisión correcta. La familiaridad debe de ser la mejor opción de diseño.

Un aspecto fundamental a tener en cuenta en el diseño de la interfaz es que existen diferentes clases de usuarios y cada uno de estos tienen unas habilidades y necesidades diferentes, por lo que es importante tenerlos a todos presentes en el diseño de la interfaz de la aplicación.

– Principiantes. Ni tan siquiera manejan bien el ratón. Necesitan ser guiados constantemente y que se les facilite al máximo las operaciones. Hay que tener en cuenta que ciertas cosas que pueden ser imprescindibles para un usuario de estas características pueden llegar a resultar molestas para usuarios intermedios o avanzados, por tanto debemos facilitar mecanismos para deshabilitar ciertas características de la aplicación exclusivamente orientadas a usuarios noveles. Por poner un ejemplo, un asistente gráfico con una figura al estilo de la que aparece en Word aconsejándote y ayudándote cuando no se le llama puede resultar muy molesto.

– Intermedios. Tienen ciertas habilidades básicas como el uso del ratón, doble clic, arrastrar y soltar. Estos usuarios buscan interactuar con el sistema de manera sencilla y rápida.

– Avanzados. Buscan cómo realizar las tareas de la forma más eficiente y rápida posible. Es por tanto, aconsejable incorporar mecanismos a la aplicación tales cómo combinaciones de teclas para atajar ciertas operaciones o poder hacer una operación de varias formas posibles siempre que la forma más complicada tenga, por lo menos, la recompensa de la rapidez.

El diseño de las interfaces debería estar en las primeras fases del ciclo de diseño de una aplicación. Muchas veces intentar mejorar la interfaz una vez terminado el producto suele ser un error. A los usuarios no les preocupa qué hay dentro de la caja (interioridades de la aplicación), siempre que haga lo que tenga que hacer. Lo que quieren son resultados y que estos sean satisfactorios. En realidad todo lo que ven y perciben es interfaz. Por tanto, en definitiva desde el punto de vista del usuario/cliente la interfaz es el producto.

En términos de interfaz el proceso de instalación y mantenimiento de la aplicación también forma parte importante a pesar de que solo se realiza una vez. Es importante que la instalación y primer uso de la aplicación sea una experiencia satisfactoria para el usuario y que desde el primer momento se sienta cómodo con el uso del producto. Cuantas menos veces se requiera de la interacción de los usuarios en el proceso de instalación y mantenimiento de la aplicación más cómodos se sentirán.

El mantenimiento y las actualizaciones deben ser totalmente transparente a los usuarios permitiendo que estos sigan trabajando normalmente con sus aplicaciones mientras éstas automáticamente se actualizan para corregir fallos o simplemente para mejorar o añadir mecanismos nuevos a la aplicación. De esta manera es importante el que las actualizaciones se hagan por la Red, con una cierta política de seguridad, para evitar al usuario que tenga que estar actualizando su software mediante soportes tradicionales como el disquete o el CD-ROM y, además, lo que es más grave, requiriendo su interacción o la de un técnico para realizar una tarea no relacionada estrictamente con su quehacer cotidiano.

Usabilidad

Es una medida de calidad sobre la facilidad de una interfaz. La palabra "usabilidad" también se refiere a los métodos para mejorar la facilidad de uso de las interfaces durante el proceso de diseño.

La usabilidad contempla cinco componentes de calidad:

– **Aprendizaje.** ¿Cuán fácil es para los usuarios llevar a cabo las tareas básicas de una aplicación la primera vez que hacen uso de ella?

– **Eficiencia.** Una vez que los usuarios han aprendido el diseño, ¿Con qué rapidez pueden llevar a cabo las tareas?

– **Facilidad de memorización.** Los usuarios vuelven a la aplicación después de un periodo de no usarlo, ¿vuelven a utilizarlo adecuadamente con facilidad?

– **Errores.** ¿Cuántos errores hacen los usuarios, cuán graves son? ¿Pueden recuperarse fácilmente de estos?

– **Satisfacción.** ¿Qué grado de satisfacción tiene el usuario al hacer uso de la aplicación?

Internacionalización y accesibilidad

Hay que tener en cuenta en fases tempranas del diseño estos aspectos fundamentales. La internacionalización se refiere al hecho de diseñar e implementar la aplicación con vistas a poder adaptar fácilmente el producto a cualquier idioma del mundo. Esta decisión es importante, si no se tiene en cuenta, posiblemente todos los mensajes, títulos de ventanas, botones etc., estarían lo que en la jerga de los programadores se conoce cómo "*hardcoded*", o sea incrustado en el código fuente, dificultando de este modo hacer una versión del mismo programa con otro idioma ya que requeriría una recompilación.

En cambio si se siguen guías de internacionalización de la aplicación posiblemente todos los recursos idiomáticos estén separados del código fuente mediante archivos de recursos siendo el proceso de traducción de la aplicación a otro idioma un proceso tan sencillo como traducir estos archivos de recursos "Clave -->valor al idioma" deseado.

La accesibilidad es un aspecto muy importante a tener en cuenta desde las primeras fases de diseño. La accesibilidad es todo aquello relacionado con el diseño de software cuyos potenciales usuarios sean todos, incluyendo aquellos con deficiencias físicas y psíquicas.

CONCLUSIONES Y RESUMEN

En este capítulo se aborda cómo la informática puede ayudar en el desarrollo de sistemas de historias clínicas electrónicas.

Tal y como fue concebido en los años 80, el concepto de sistema centralizado ya no existe. La complejidad de los entornos sanitarios hace que convivan numerosos sistemas de información en un entorno distribuido que obligan a establecer un control riguroso para poder prevenir problemas de incompatibilidad de la información.

En el mundo de los sistemas distribuidos ha habido, desde los inicios, numerosas tendencias y arquitecturas: cliente-servidor, multicapa, servidores de aplicaciones, Grid, *Web Services*, etc. Parece ser que las que actualmente están más en boga son las aplicaciones multicapa con las propuestas tecnológicas de J2EE y .NET junto a los *Web Services* y CORBA.

Como propuesta más novedosa y que parece tener bastante apoyo por parte de la industria e investigadores está la tecnología Grid.

Los *Web Services* pueden solucionar fácilmente problemas de interoperabilidad e intercambio de mensajes e información entre aplicaciones en una organización para solucionar los problemas de desintegración. La tecnología GRID también puede solucionar esto, pero requiere un esfuerzo mayor que recompensa si se van a utilizar otras de las numerosa características y virtudes del Grid Computing, como el aprovechamiento de la potencia de cálculo de todos los recursos en los tiempos muertos de procesador para realizar una tarea con grandes necesidades de cálculo o creación de organizaciones virtuales, integración de recursos distribuidos, computación universal, etc.

En cualquier caso es necesaria la integración de datos. Integrar es combinar datos de forma que puedan ser compartidos, para alcanzar este fin existen distintos métodos de integración que van desde la utilización de mensajes de intercambio de información entre aplicaciones hasta la construcción de un *Data Warehouse*.

Un *Data Warehouse* se diseña con el propósito de almacenar y consultar grandes cantidades de información relacionada. La información proviene de diversas fuentes, por tanto, un *Data Warehouse* proporciona una vista unificada y consistente de información proveniente de diversas fuentes. Además, proporciona herramientas para la toma de decisiones, ejecución de consultas y generación de informes.

Cuando por alguna circunstancia y después de planteados todos los problemas no queda más remedio que plantear y justificar una migración del sistema hay que definir todas las estrategias necesarias con el fin de minimizar los problemas que se puedan generar, incluyendo un mayor número de herramientas que faciliten y ayuden en todos los procesos necesarios para llevar una migración con éxito.

En casi todos los casos es posible y sobre todo preferible comunicar y consultar información entre distintos sistemas mediante el intercambio de documentos. XML parece ser la herramienta ideal para realizar estos procesos, con el consiguiente aprovechamiento de las ventajas que proporciona, por ejemplo la utilización de los *Web Services*. Por otro lado las ontologías pueden ser la herramienta ideal para describir formalmente fuentes de información clínica.

Por otra parte, cada vez se hace más evidente la necesidad de un diseño eficaz de los sistemas con respecto a la interfaz utilizada para la comunicación con el ser humano. Los Analistas de Sistemas, no solo deben considerar qué debe hacer el sistema y cómo (computacionalmente) va a realizarlo, sino también considerar los principios de percepción y ergonomía que afectan al ser humano, para lograr un producto que cumpla con los requerimientos y que se adapte a los usuarios del mismo.

Por tanto, es importante no descuidar los aspectos de interfaz, usabilidad, ergonomía etc. de una aplicación. Es más hay que tenerlos presentes desde las primeras fases de desarrollo. En este sentido no hay que escatimar en la inversión necesaria para este fin, de hecho, los expertos estiman que debería ser del 10 del costo total del proyecto.

También se ha de tener presente que la aplicación debe trabajar correctamente pues por muy buena interfaz que se tenga, el usuario lo que quiere es que el programa haga lo que tenga que hacer y además de forma correcta. Hay que buscar la simplicidad y claridad en los diseños y que estos se ajusten a las diferentes capacidades y conocimientos sobre manejo de ordenadores de los usuarios.

ENLACES DE INTERÉS

- Java 2 platform, Enterprise edition (J2EE). Disponible en: http://java.sun.com/j2ee/.
- Plataforma.NET. Disponible en: http://www.microsoft.com/net/.
- Object Management Group. Disponible en: http://www.omg.org/.
- CORBA. Disponible en: http://www.corba.org/.
- CorbaMed Disponible en: http://healthcare.omg.org/.
- Editor ontologías protégé: http://protege.stanford.edu/index.html.
- Proyecto europeo para el intercambio de información basado en ontologías para la gestión del conocimiento y elcomercio electrónico: http://www.ontoweb.org.
- Grid Computing Info Centre. Disponible en: http://www.gridcomputing.com/.
- Global Grid Forum. Disponible en: http://www.gridforum.org.
- Open Grid Services Infrastructure Working Group. Disponible en: https://forge.gridforum.org/projects/ogsi-wg.

- Open Grid Services Architecture Working Group. Disponible en: https://forge.gridforum.org/projects/ogsa-wg.
- Legacy information systems Issues and directions. Disponible en: http://www.comp.dit.ie/bwu/IEEESoftware16_5.pdf.
- Portal del Comité Técnico 251 del Comité Europeo de Normalización (CEN/TC251). Disponible en http://www.centc251.org.
- Portal del W3C en relación a XML, donde se puede encontrar toda la información sobre XML y tecnologías asociadas. Disponible en: http://www.w3.org/XML.
- W3Schools - Full Web Building Tutorials - All Free: En este portal se puede encontrar numerosos cursos sobre XML. Disponibles en http://www.w3schools.com/xml/default.asp.
- W3C. Web Services Activity. Disponible en: http://www.w3.org/2002/ws/.
- Web Sevices Management. Disponible en: http://www.webservices.org.
- The simple object Access Protocol and an Online Health information Infrastructure. Disponible en: http://www.informatics-review.com/thoughts/soap.html.
- Object Management Group. Disponible en: http://www.omg.org.
- Official Guidelines for User Interface Developers and Designers. Disponible en: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwue/html/welcome.asp.
- Sun Java UI guidelines. Disponible en: http://java.sun.com/products/jlf/ed2/book/index.html.
- Web UI guidelines. Disponible en: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsiteplan/html/improvingsiteusa.asp.
- Government of Canadá Internet guide. Disponibles en: http://www.cio-dpi.gc.ca/ig-gi/.
- Jakob Nielsen's Website. Disponible en: http://www.useit.com/.

BIBLIOGRAFÍA

- Bernstein, P., Middleware, Communications of the ACM 1996; 39 (2).
- Bisbal J., Wu B., Lawless D., Grimson J. Building Consistent Sample Databases to Support Information System Evolution and Migration. Database

and Expert Systems Applications (DEXA98), Lecture Notes in Computer Science 1460, Berlin. Springer Verlag; 1998, p. 196-205.

– Bouguettaya A., Benatallah B., Elmagarmid A. Interconnecting heterogeneous information systems. Kluwer Academia Publishers; 1998.

– Brodie M., Stonebraker M. Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach. Morgan Kaufmann Publishers, 1995.

– Coulouris G., Dollimore J., Kindberg T. Distributed Systems: Concepts and Design. 3rd Edition. Addison-Wesley; 2001.

– Fensel, D. Ontologies: a Silver Bullet for Knowledge Management and Electronic Commerce. 2nd Edition. Springer, 2004.

– Gómez-Pérez, A., Corcho, O. Ontology languages for the Semantic Web. IEEE Intelligent System; 2002, 17(1), pp. 54-60.

– Grimson J. Stephens G. Jung BN. Grimson W. Berry D. Pardon S. Sharing Health-Care Records over the Internet. IEEE Internet Computing 2001; 3: 49-59.

– Gruber, T. R. A translation approach to portable ontology specifications. Knowledge Acqusition, 1993, vol 5, pp. 199-220.

– Gutiérrez Rodríguez A. XML a través de ejemplos de publicado. RA-MA; 2001.

– Inmon W. H. Building the Data Warehouse. 2ª ed. Wiley Computer Publishing; 1996.

– Kay S. Architecture models to facilitate communication of clinical information. Methods Inf Med 1999; 38:326-31.

– Livingstone G. Guía Esencial de XML. Prentice. 2002.

– McGinness, D. L., Fikes, R., Rice, J., Wilder, S. The Chimaera ontology environment. In the Proceedings of the 17th National Conference on Artifical Intelligence, 2000.

– Monday P. B. Web Service Patterns: Java Edition. The Author Press; 2003.

– Newcomer E. Understanding Web Services: XML, WSDL, SOAP, and UDDI. Independent Technologies Guide. Addison Wesley (Paperback); 2002.

– Norman D. A. The Design of Everyday Things. Basic Books. Perseus; 2002.

- Preece J., Rogers Y., Sharp H. Human-Computer Interaction. Addison-Wesley; 1994.
- Raskin J. The humane interface. Addison-Wesley; 2000.
- Rector, A., Solomon, W., Nowlan, W., Rush, T. A Terminology Server for Medical Language and Medical Information Systems. Methods of Information in Medicine: 1997. Vol. 34, pp. 147-157.
- Rusty Harold E. XML Bible. 2ª ed. Publicado por Hungry Minds; 2001.
- Scribner K., Stiver M. C. Understanding SOAP. SAMS; 2000.
- Tanenbaum A. S. Sistemas Operativos Distribuidos. México. Prentice Hall Hispanoamericana, S. A.; 1996.
- Thompson, J. Avoiding a middleware muddle. IEEE Software; 1997. 15 (6), pp. 92-95.
- Van Bemmel J. H., Musen M. A. (eds). Handbook of Medical Informatics. Heidelberg: Springer-Verlag; 1997.
- Wu B., Lawless D., Bisbal J., Grimson J., Richardson R. The Butterfly Methodology: A Gateway-free Approach for Migrating Legacy Information Systems. Proceedings 3rd IEEE Conference on Engineering of Complex Computer Systems; 1997. pp. 200-205.