A UML-based Architectural Framework for Web-based Information Systems

Egil P.Andersen
Norwegian Computing Center
P.O.Box 114 Blindern, 0314 Oslo, Norway
Tel: +47 22 85 25 94, Fax: +47 22 69 76 60
Egil.Paulin.Andersen@nr.no

Abstract

Web-based information systems essentially provide access to information from "anywhere" to "anyone". The SynEx project aims to provide any authorised healthcare professional with integrated access to electronic patient records where different record parts may reside in different information systems, nationally or internationally. An important challenge is thus how to provide a seamless integration of possibly globally distributed information. Furthermore, information systems like this must be generic and extendable in the sense that the kind of information they must be able to manage is not fully known at design-time. They will evolve over time, and they must be designed for this. The SINAI project aims to demonstrate that a standardised use of UML in combination with XML can make it easier to achieve integration and interoperability of distributed information and applications, and model and component integration techniques can be used to support development of generic and extendable web-based information systems.

Keywords

Information systems, distributed systems, systems integration, internet, UML, XML

A UML-based Architectural Framework for Web-based Information Systems

Egil P.Andersen
Norwegian Computing Center
P.O.Box 114 Blindern, 0314 Oslo, Norway
Egil.Paulin.Andersen@nr.no

Abstract

Web-based information systems essentially provide access to information from "anywhere" to "anyone". The SynEx project aims to provide any authorised healthcare professional with integrated access to electronic patient records where different record parts may reside in different information systems, nationally or internationally. An important challenge is thus how to provide a seamless integration of possibly globally distributed information. Furthermore, information systems like this must be generic and extendable in the sense that the kind of information they must be able to manage is not fully known at design-time. They will evolve over time, and they must be designed for this. The SINAI project aims to demonstrate that a standardised use of UML in combination with XML can make it easier to achieve integration and interoperability of distributed information and applications, and model and component integration techniques can be used to support development of generic and extendable web-based information systems.

1. Distributed Information Systems for the Internet

Distributed information systems play a key role in the business of many companies, organizations and institutions. They may rely on an ability to manage and maintain business information from several heterogenous information sources, where authorised users can access and share this information with few constraints as to where they are located geographically, and sometimes also the equipment they are required to possess.

Distributed information systems are typically characterised by a layered architecture from relational and/or object-oriented databases at the bottom, to possibly ultra-light WAP based clients at the user end. The design and implementation of large-scale distributed systems greatly benefits from standardisation efforts in UML, to ease analysis/design collaboration and communication between distributed development groups, repository models for the management of meta-information, software component technology as (D)COM and CORBA for technical interoperability and support for more open and extendible software architectures, XML for transmission of semantically meaningful information, and web and WAP/WML browsers as readily available presentation software.

SynEx - Synergy on the Extranet

Siemens Health Services (SHS) develops amongst others information systems for electronic patient records (EPR). In order to support the continuity of care between organisations, and make relevant patient information readily available, there is a growing trend towards *shared care*, which require different EPR systems to be able to share their data. This has led to projects like *Synapses* [6][7], for EPR standardisation, and its follow-up *SynEx*[1] [1][2][3][4][5] which aims to provide healthcare professionals with integrated access to patient records and related information, despite that different record parts may reside in different EPR information systems, nationally or internationally. That is, the ultimate goal for electronic patient records is to provide *complete* access to *any* part of *any* record to *any authorised* healthcare professional in a *secure* way.

The results from SynEx shows that this is likely to be an achievable goal for production-use EPR systems. However, there are also other important challenges for EPR systems. Healthcare information systems, and EPR systems in particular, are *open and generic* information systems in the sense that the kind of information they must be able to manage is not fully known at design-time. Electronic patient records are very diverse, and it is difficult to standardise them to a desirable level of detail (even among hospitals within the same city!). They will evolve over time, and they must be designed for this.

SINAI - Seamless Integration of Non-homogenous Applications and their Information

We believe that the above challenges of EPR systems also apply to a number of other domains, e.g. information systems for finance, government, education, and more; see e.g. [8]. Thus, based on experience from SynEx, and other projects with similar characteristics, the Norwegian Computing Center has funded (until mid-spring 2001)

[1] SynEx is a major EU (European Union) Health Telematics project, from 7/1998 until 9/2000.

the SINAI project which defines a UML-based architectural framework for web-based information systems. SINAI is an architectural framework that supports the development of *generic* and *extendable* distributed information systems on the internet/intranet/extranet, that allow for *domain-specific* specialisation, customisation and optimisation. Other SINAI architectural characteristics are support for seamless integration of information from heterogenous data sources and legacy systems, seamless integration of independent applications, platform independence and rapid prototyping.

Before considering SINAI in further detail we will first provide an outline of the SynEx architecture. This is fairly "mainstream" with respect to the latest in internet technology[2], but useful as an introduction to the main constituents of web-based information systems whose architecture will benefit from SINAI.

2. SynEx - Seamless Integration of Distributed Electronic Patient Records

Figure 1 illustrates the layered architecture that has been designed and implemented by SHS in SynEx to demonstrate, for proof of concept, integration of distributed electronic patient records. There is a *client presentation and interaction layer*, a *web server*, an *application layer*, a *data independence layer*, and a *data layer*.

The Oslo EPR server is implemented on a Microsoft platform, but most of the issues that concern our layered server-side architecture, and SINAI as described below, also applies to other platforms; e.g. involving Apache web servers, EJB (Enterprise Java Beans), Java Servlets, JDBC, Oracle, and so on.

SynExML - XML for Exchanging Healthcare Record Information

XML [9] has the power to become the independent data exchange format of the future. The use of XML to exchange data between heterogenous systems provides support for hierarchically structured data, user defined tags and machine-understandable assertions for searching, reasoning and analysing information like electronic patient records.

An XML DTD, called the *SynExML (SynEx Markup Language)*, has been defined within the SynEx project to be used for inter-site exchange of EPR information. That is, SynExML is the basis for semantic interoperability between SynEx components that relate to EPR information. SynExML is based upon the generic EPR structure defined by the Synapses EPR Server specification [6][7].

Clients

An information system for the internet/intranet/extranet will typically serve a number of different types of clients; e.g.

- *"thin" clients* - web browsers without other software than what is downloaded upon request.
- *"fat" clients* - with a number of application specific components preinstalled.
- *mobile clients*[3] - e.g. a WAP based client with a WML browser[4].

For reasons described below (XML+http=SOAP), all these clients can be made independent of the server-side technology, and they can all log on to a server and then retrieve and update authorised information. The main difference between "thin" and mobile clients versus "fat" clients, e.g. the SynEx client, is that the latter are clients dedicated to their use; e.g. a hospital desktop computer. They would typically include e.g. extensive caching, and flexible options for checking in and out information and then work on this information offline before updating the server with changes made, etc. "Thin" clients typically include e.g. private home PC's.

Figure 1 also illustrates the SynEx Client, a prototypical "fat" client component architecture. It is like a small-scale version of the server-side architecture. The application specific components[5] are made for a particular application that uses EPRs stored on SynEx EPR servers, while the application independent components are made for any application that needs to access such records. The latter are divided into three different kinds of (COM) components. The cache stores in memory retrieved records, and beside for performance reasons, this makes it possible to work with records in offline mode. The cache can also be saved to local files and restored later on. The "data provider" components are responsible for communication with different kinds of web servers; e.g. connecting to the Oslo server (ASP/XML) is different from connecting to the Dublin server (CGI-scripts). In addition a separate provider is made for offline access to local storage. Finally

[2] on a Microsoft platform for the SynEx demonstrator. SINAI is not platform dependent, but a first demonstrator version will also be implemented for a Microsoft NT/Win2000 platform.

[3] No mobile client support has been implemented in the current version of the SynEx demonstrators.

[4] in a few years, as mobile networks and client devices become more powerful, there may not be much that distinguishes mobile from stationary clients.

[5] either duplicating or complementing server side application layer components

there is a set of XML parser components; one for each different kind of XML that can be received (SynExML formatted XML is one of them).

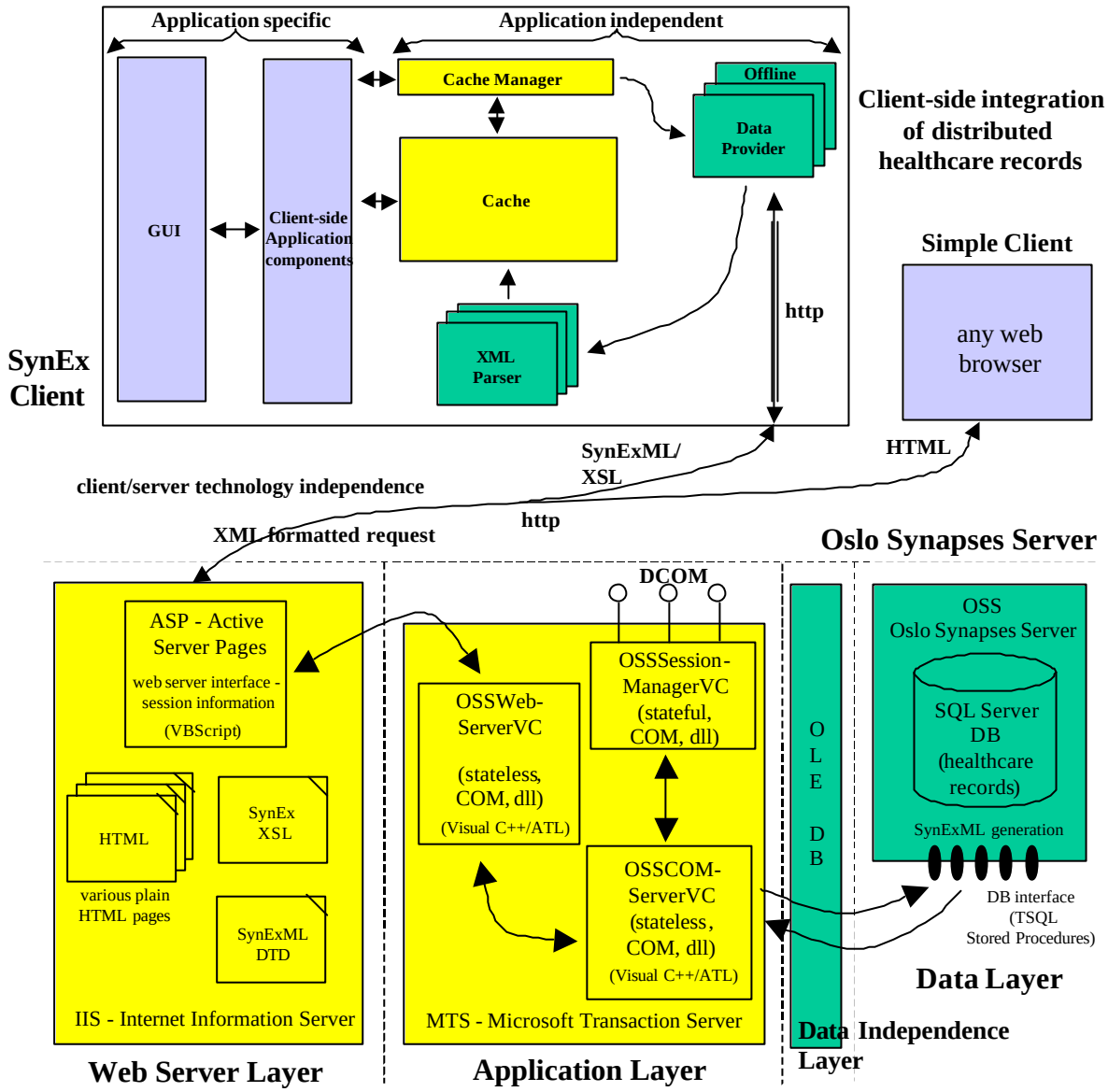

Figure 1. *The technical architecture for the SHS SynEx demonstrator.*

SOAP - XML over http for Request/Response Interaction and Information Transmission

Component technology like e.g. *(D)COM* [20] and *CORBA* [12] represents a major achievement towards the goals of creating extendible and adaptable software architectures by supporting *technical interoperability* between objects independent of in which programming language these objects are implemented. DCOM and CORBA also supports *location independence*, i.e., independence of where programs execute, to a certain degree, but this has its limitations.

SOAP (Simple Object Access Protocol) [10][11] is an XML format (DTD) for client-server communication over http where both client requests and server responses are formatted as XML. SOAP may become a W3C [9] standard for such use of XML.

There are several advantages by using XML over http for client-server interaction despite http's functional simplicity relative to e.g. DCOM or IIOP (CORBA). http is a simple protocol with good coverage and few demands on the client, and most firewalls are readily configured for common security options dealing with well known internet protocols and ports. This as opposed to DCOM and IIOP for which firewalls can pose a problem. In practice, DCOM and IIOP can be well-suited for computers within e.g. a limited area, but not between "any" remote client and server on the internet. XML also has the great benefit that it makes the underlying client- and server-side technology transparent to each other. For example, the Oslo EPR server is implemented on a

Microsoft platform, while the Dublin EPR server uses an Apache server with CGI scripts. Thus similar to how component technology like COM provides for programming language independence and technical interoperability locally, SOAP provides for platform independence and technical interoperability globally. Furthermore, XML is likely to be well-suited for mobile clients, e.g. WML is a particular XML format, and there are already numerous software tools and systems with good support for XML.

Web Server

The Oslo EPR server uses *IIS (Internet Information Server)* [22] as its web server, and IIS uses *ASP (Active Server Pages)* objects and scripts as its external client interface. For performance reasons, amongst others, the web server layer should preferably be as "thin" as possible. This means that immediately after an ASP script receives an XML request string from the client, it just hands this XML string over to a COM component (*OSSWebServerVC*) in the transaction server for further processing, and returning results to the client.

Client Requests

XML formatted client requests can be sent to the server in a number of ways. One alternative is a http GET command with a "querystring"[6] (which for several reasons should *only* be used for demonstration purposes); e.g.

```
http://citroen.nr.no/sia/oss.asp?<OSSrequest><Function Name="RecordInfo">
   <Arg Name="User">onordmann</Arg><Arg Name="RecordID">{C7910C91....0000}</Arg>
   <Arg Name="Retrieval">shape</Arg><Arg Name="ResponseType">html[7]</Arg>
   </Function></OSSrequest>
```

Other alternatives are http POST commands in HTML form's (with the same XML syntax). On a Microsoft client then the *XMLHttpRequest* ActiveX control can be very useful for sending and receiving XML in a SOAP manner (this is used by our "fat" SynEx client).

Application Layer

An application layer consists of a set of software components, e.g. *COM* components [20], that execute within a *transaction server*. Transaction servers are important for *scalability* with respect to the number of concurrent users, and thus *performance*, and also for managing *distributed transactions* and *resources* like database connections.

For the Oslo EPR server we use the transaction server from Microsoft, *MTS (Microsoft Transaction Server)*, which amongst others support *distributed transactions*, *database connection pooling*, and (from v.3) *object pooling*. The support for distributed transactions means that COM objects residing in the MTS on different computers can participate in the same atomic transaction. Similarly, if an MTS COM object works on several databases on different computers then these database operations can be combined into a single transaction.

Database connection pooling means that instead of assigning a dedicated database connection to each client, a pool of database connections are reused as and when required to serve client requests. The benefit is that database connections, which are a limited resource, can be utilised more efficiently such that the number of concurrent users can exceed the number of concurrent database connections, and the (severe) performance overhead of creating database connections can be reduced.

Object pooling means that instead of creating an object instantiated from a particular component from scratch each time it is needed, which can also be time-consuming, objects that are currently "free", i.e., not participating in a particular transaction initiated by a client, can be reused *as if* they were newly created objects. However, COM objects used in object pooling must be made *"stateless"*; i.e., they may well have state when they participate in transactions, but after a transaction is ended, and the object returns to its pool of now ready objects, then no one else should depend on its state since as soon as it is assigned to a new transaction, its previous state will be overwritten as if it were a newly created object (see the "application models" in section 3).

[6] Notice that the SynEx demonstrator complies with SOAP technically, but not the SOAP XML DTD for client requests (due to an unfinished standard at the time of implementation).

[7] If the client specifies *"html"* for the "ResponseType" argument, *or* no "ResponseType" argument is provided *and* the client uses any other browser than Internet Explorer v.5.0, then the XML generated for the information requested will be transformed into HTML on the server-side via the use of an XSL specification. Requesting HTML instead of XML is mostly relevant in those cases where the client only wants to browse the information received, and the browser is unable to transform XML into e.g. HTML via XSL; either the default XSL provided from the server, or some other XSL that the client has access to.

Data Layer

Our data layer consists of a set of data sources that contain persistent health information. These data sources may include e.g. plain files, or the like, but primarily they will be databases like e.g. SQL Server[19]. Plain text files like e.g. HTML, XML or WML files may be made available to client users for efficient read-only access, but such files should be generated "on demand" based on information in databases. Files are generally not very suitable as the original source of information. Except for reading/browsing, then .html/.xml/.wml/etc files are highly proprietary, and the availability of their content is not satisfactory since their information content cannot be reorganised or dynamically integrated in new ways. Hard-coded hyperlinks between such files are also hard to maintain.

The Oslo EPR server database provides an external interface of *Transact-SQL stored procedures*, and such stored procedures are used for generating the SynExML that is returned to the client. There are several good reasons for always encapsulating database tables behind an "interface" of stored procedures: 1) constraints and business rules that are inherently linked to the database schema, independent of which application is using the database, should be enforced within the database; 2) the tables of a database schema are often subject to minor changes, e.g. for performance reasons, and such changes should be transparent to the application layer; 3) executing stored procedures is often more efficient than executing SQL statements requested by application components individually.

Data and Application Layer Independence

OLE DB [21] is a key part of Microsoft's strategy (*Microsoft Universal Data Access (UDA)*) to enable seamless access from an application layer to several heterogenous information sources; e.g. databases, directory services, file systems, etc. OLE DB itself is a set of COM interfaces that are implemented by *data providers* which are objects that offer information from e.g. a database, a file, etc, and *data consumers* which are objects that make requests for certain information from one or more data providers.

ADO is a simplified version of OLE DB primarily made to make its functionality available also to scripting languages, but ADO can also be well-suited for information caches in "fat" clients like our SynEx Client.

Customising Client Information Presentation

XML in combination with *XSL (eXtensible Stylesheet Language)* makes it possible to provide many different kinds of client-side presentations of the same XML formatted information. For example, the same XML string can via XSL be presented as WML in the WML browser of a mobile phone, as one kind of HTML in Netscape, as a slightly different HTML in Internet Explorer, as computer generated speech for a blind person, and so on.

For Internet Explorer clients then DHTML[8] (Dynamic HTML) can be used as an alternative to XSL for information presentation. The advantage of DHTML over XSL is that DHTML is more explicit when it comes to organising the presentation layout.

Our SynEx client supports three different means for presenting information within its Internet Explorer container, namely XSL, DHTML and ActiveX controls. By editing or creating new XSL specifications, DHTML pages, or ActiveX controls within HTML pages, any authorised user can fully control and customise client presentations without changing or recompiling any other client components. However, particularly for generic data structures like the Synapses EPR specification, and thus SynExML formatted XML, it is quite time-consuming to create high quality XSL specifications. Thus while XSL and DHTML may well be used for certain ad-hoc, on-the-fly presentations, for dedicated clients then ActiveX controls or Java applets are better suited for high quality graphical user interfaces.

Distribution and Integration

As illustrated in figure 2, we can distinguish between different kinds of distribution in an information system architecture, namely *client-*, *application- and database distribution*.

Database distribution utilises the e.g. SQL Server distribution features, while distribution at the application layer is handled by e.g. MTS as a collaboration of MTS objects that reside on different computers.

By *"client distribution"* is here meant that distributed information is integrated at the client level, i.e., as the result of a client's request for related but distributed information (e.g. different parts of the same EPR). The application and data layers are not involved in the distribution except that they may contain information on how and where to get access to related information that resides elsewhere.

Database distribution should be transparent to the application layer, while distribution at the application layer should be transparent to its clients. Typically, database and application layer distribution is "local" distribution in

[8] DHTML is a technique where an HTML page can be dynamically updated, e.g. display new data or change page layout, while it is displayed in a browser and without reloading the page (no "flicker").

the sense that the databases or application components exist on computers in physical proximity, or at least the same development organisation is in charge of configuring each participant in the distribution.

Client side integration may well be an integration of globally distributed information where the organisation offering the server side functionality and information may have no knowledge of who are the clients, and the client devices may well connect to the server from anywhere in the world.

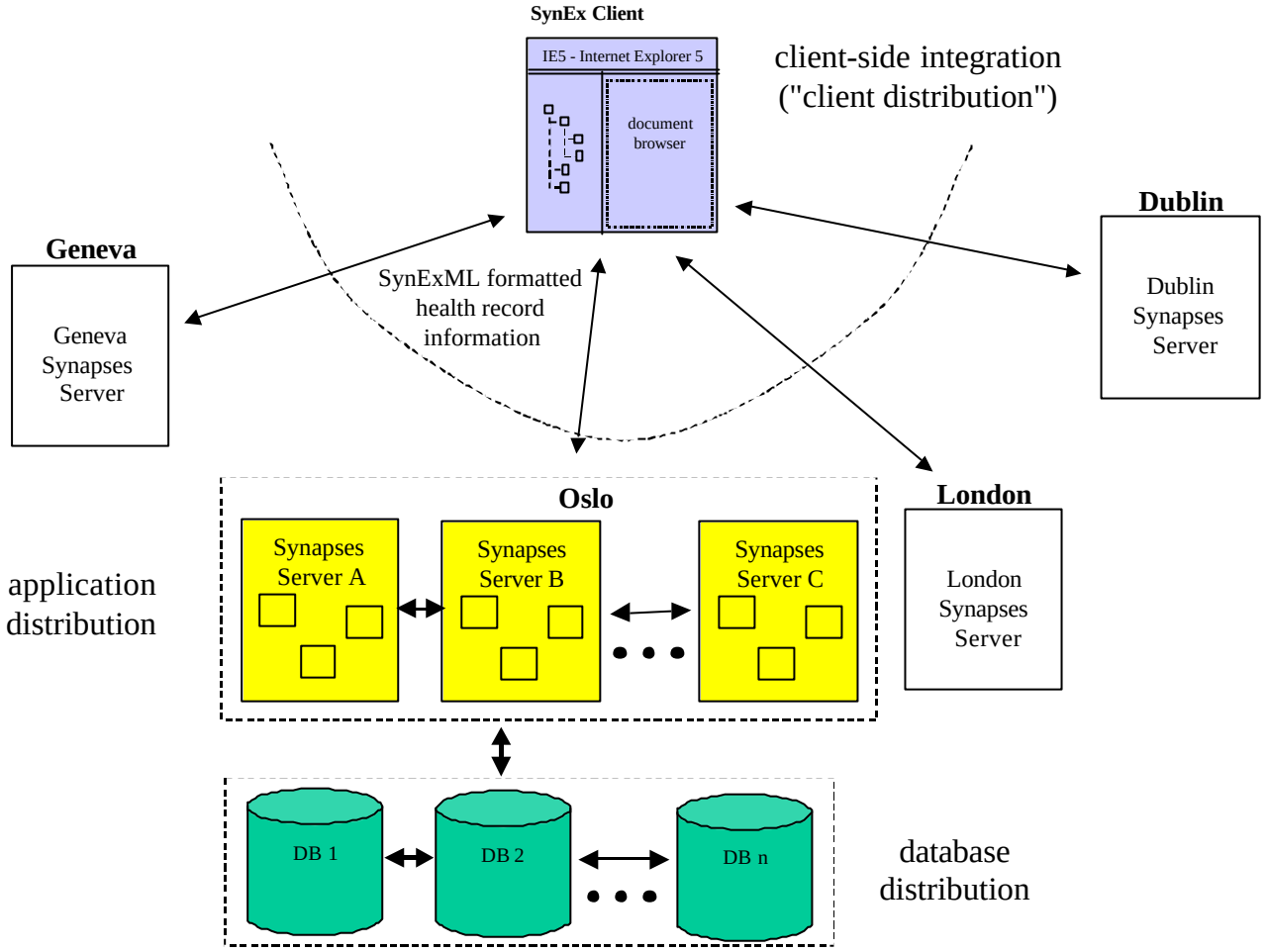

Figure 2. *Client versus Application versus Database Distribution.*

The Synapses EPR Server specification of SynEx does not rely on a central catalog service for retrieving information on where various parts of a particular EPR resides. Instead this information is distributed such that every record on a particular server has the information required to access other parts of it that exists on other servers. Thus Synapses records are well-suited for this kind of client side integration since *we do not foresee any need for client-side transactions to span more than those parts of a record that reside in a single server*. That is, this is a constraint that in our experience will not severely hamper users work with EPR information, but it greatly simplifies the transaction handling of globally distributed records. We believe this to be the case for many other kinds of distributed information as well. Otherwise elaborate 2-phase commit protocols for remote clients will be required.

Long-Term Distribution - an Open Issue

The content of an electronic patient record is expected to be available "forever". The latter implies an important challenge, also from a technical perspective, that has not been addressed within SynEx. Remote record links are currently "hard-coded" in the sense that they contain a particular web server address, and in addition enough information to uniquely identity the relevant record parts on this server. However, over time web servers, databases, access rights are moved, renamed, changed, etc. Thus a more robust, long-term solution will be required to assure that distribution targets remain available.

3. The SINAI approach to Information and Application Integration

What is the role of XML?

XML can play a valuable role for transmission of information server-to-client, client-to-server and server-to-server. Its essential characteristic is that, on one hand, it is a string of text formatted according to certain rules (well-formed and possibly validated against a schema) that can easily be transferred e.g. over http from a server to a client. On the other hand, when being created and when being received by the client it can be accessed and operated on as an object-structure with an interface of functions, including events, as any other e.g. COM object; i.e., XML can be "morphed" from a string of text into a structure of objects, and vice versa. That latter is possible via the *DOM (Document Object Model)* offered by XML parsers.

Information models and object models play key roles in an information system architecture, but one lesson learned from SynEx is that despite that there are languages like DTD's and others for defining XML schemas, XML is *not* a modelling language and therefore *should not be used as such*. For example, it was very time-consuming process in SynEx for every site involved to agree on a common XML format. Many changes were made along the way, and more are to be expected, and for each change XML parsing code had to be changed.

This is a general problem, and may pose a serious obstacle to the use of XML as a vehicle for integration of distributed information sources. If every small group or business sets out to define its own XML schema, this will lead to an "explosion" in the number of non-interoperable schemas and standardisation will be jeopardised. Making changes in XML parsing code can be expensive; i.e., it is too much to expect that XML schema specific code will be fully encapsulated so that changes due to future standardisation can be confined and implemented with little cost.

BizTalk [16], OASIS [17] and XML.ORG [18] are all initiatives to support XML *schema repositories* where organisations can publish their XML schemas for the purpose of sharing XML formats and develop industry standards. However, we believe that it is possible to go one step further and consider XML as a purely technical means for communication and integration; i.e., its role is at a "lower level" in information system architectures. Instead proper modelling methods should be used for information modelling and to decide the (XML) format for transmission between clients and servers.

Standardised use of XML based on UML

UML (Unified Modelling Language) [13], and tools like e.g. Rational Rose [14], is becoming much of a standard for object-oriented and relational modelling, analysis and design. An important benefit of using UML is for communication purposes between software developers, and also between software developers and domain experts since it is relatively easy to enable domain experts to understand what is expressed by a UML model.

In order for a client and a server to interact and communicate in a meaningful way they must have a *common understanding* regarding which requests can be made by the client, and what kind of information will be returned from the server. A more general solution is to use UML models to describe which requests are available for clients, and what is the structure of the information returned. Thus in SINAI the XML that is transferred between clients and servers is based on just a single XML schema, with two major sub-schemes[9], namely schemes for

- information on object instances of a UML model[10] (actual object relationships, attribute values, etc). This also include UML model information (meta-information - classes, structural and behavioural properties, etc) since information on a UML model is no principally different from other information.
- how to invoke object operations (method/function calls - according to a UML model)

The common understanding necessary between a client and a server, communicating via XML over http, will be based on meta-information about UML models. A client will not only receive information on a structure of objects, but also meta-information on their business rules, which operations can be executed on these objects, etc. Client requests correspond to methods on various UML classes, and the information returned will be information on a particular set of objects instantiated from particular classes in a UML model. Application developers will also be presented with object models and object interfaces that all adhere to certain commonly agreed (SINAI defined) UML conventions.

An important benefit of this is that regardless of which model changes and extensions are later made, e.g. in an EPR specification, the foundation for the common understanding necessary between a client and a server will remain the same, and the infrastructure for client-server communication will remain unchanged. Of course, this is more of a syntactic foundation than a semantic foundation, but the domain specific semantics of concepts must be agreed regardless.

[9] and a few more for increments like e.g. dynamic client refresh.

[10] Notice that this as opposed to current techniques where XML formats are based on transformations of UML models to XML DTDs, and vice versa.

There is also a benefit from an information and application integration perspective here since all SINAI compliant systems use models that adhere to the same conventions regarding their structural and behavioural properties and interfaces. This both when manifested as XML or IDL (component interfaces), but at the same time each of these models are domain specific, e.g. modelling a particular part of an EPR.

Code Generation - Standardised Architectures

Due to the above, XML parsers on both the client- and server side can be implemented once and for all[11]. In addition, the UML models can be used to *generate* parts of the model specific code in server- and ("fat") client side components. A benefit from this, beside saving some coding, is also a more standardised architecture (at least for a particular product line). Figure 3 illustrates which kind of code it is relevant to generate relative to a layered architecture as illustrated in figure 1.

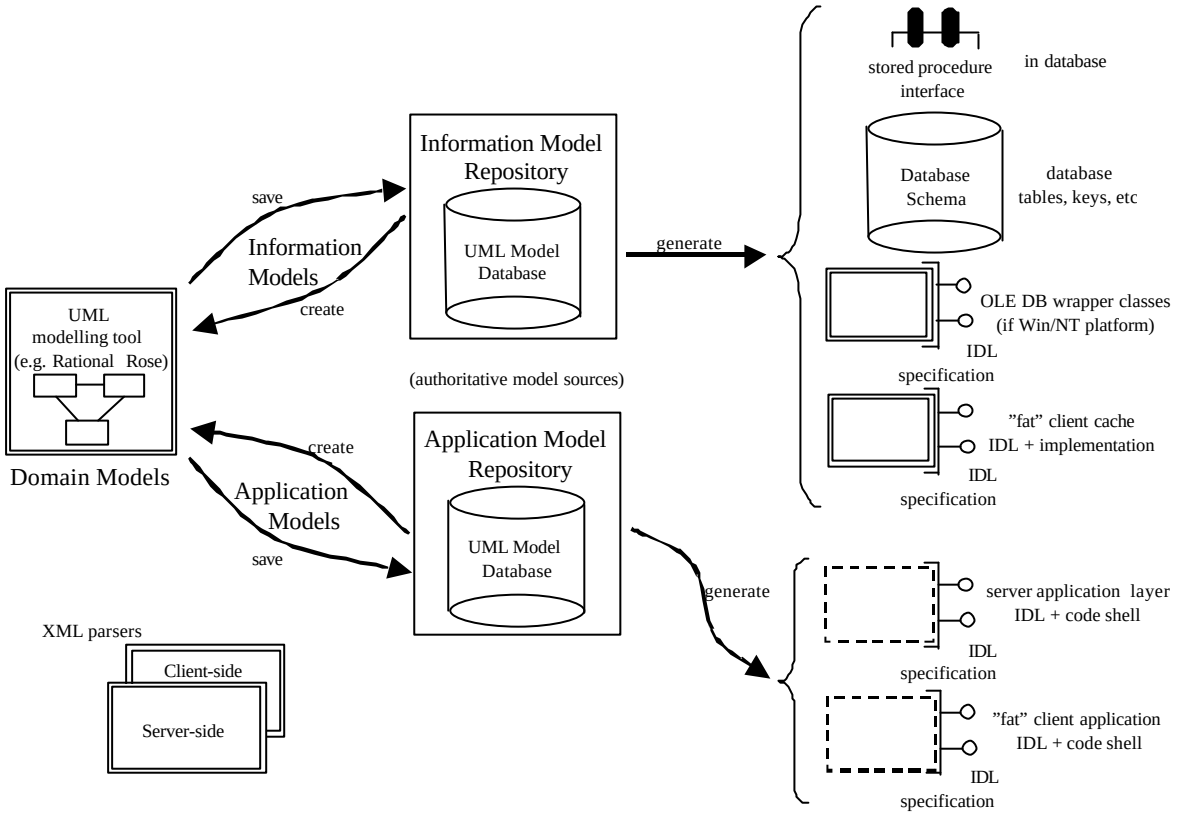

Figure 3. *Auto-generated code in SINAI*

In SINAI we make an explicit distinction between *information models*[12] versus *application models*.

Information models define (stateful) objects residing on persistent storage (primarily databases), and their interfaces contain functions for creating, removing, updating and retrieving objects (or actually, "role instances" - see below). The data layer implementation of these functions handle information specific, but application independent, constraints.

Application models are service-oriented (e.g. originating from Use Cases) in the sense that each model focuses on a particular set of related services (sub-application) that can be applied to particular information models. The application layer objects that implement application models are mostly *stateless objects* as required for scalability and good performance in large-scale information systems; cf. "Application Layer" in section 2.

Both information and application models are made for a particular domain, and they are both defined in e.g. Rational Rose, but they are each assigned a globally unique identifier (GUID) and kept in a model repository as the authoritative source for model meta-data. Authorised clients will have access to this information the same way they will have access to objects implementing these models, and repositories can be distributed similar to how object instance databases are distributed.

From the model repository, an information model can be used to generate a database schema for object instances, a stored procedure interface for this schema, possibly OLE DB wrapper classes (for Win/NT

[11] They only need to change if UML, or conventions for its use, change.

[12] The distinction in [15] between *"implementation models"* versus *"interface models"* only applies to SINAI information models.

platforms), and "fat" client cache components (both their IDL and also a cache implementation). An application model cannot, in general, be used to generate actual application code, but it can be used to generate IDL specifications, and possibly code shells for their implementation. This both for application components in the server side application layer, and also for "fat" clients if needed.

The "automatic" use of generated code should not be overestimated. Manual customisation and specialisation must be possible at all levels and all layers of the architecture. For example, complex business rules usually require non-trivial manual intervention. The main challenge is the organisation of predefined code versus automatically generated code versus case-specific manually made code, and this is were component technology provide a great benefit with its distinction between interfaces versus implementation, and the ability to change encapsulated components independently.

Generic and Extendable Information Systems

Healthcare information systems, and EPR systems in particular, must be *generic* and *extendable* in the sense that the kind of information they must be able to manage is not fully known at design-time. They will evolve over time, and they must be designed for this. Thus they must support evolutionary development and seamless integration of separately developed subsystems and/or legacy systems.

Possible solutions to this are, amongst others, using a very generic database schema and application layer, with support for specialisation and customisation. This is the SHS Synapses EPR server approach. Another alternative is using a very generic application layer that can be applied to "any" UML Class Diagram; see [8]. Genericity provides flexibility, but both these solutions have some disadvantages due to their generic nature. In the former at the cost of added complexity, and in the latter case, performance and it is difficult to support manually made customisations.

A SINAI based system will have many of the characteristics of a generic system, but without being generic itself; i.e., it will not predominantly consist of generic software components. Instead it will consist of dynamically composable units of information, and dynamically configurable application components. More specifically, a SINAI server data layer is defined by a set of UML information models that each offer *roles* (as opposed to traditional UML classes) that can be instantiated into role instances and dynamically composed into persistent, stateful information objects ("subjects" according to the conceptual role model of [23]); as illustrated to the left in figure 4. Information objects can "play" different roles during their lifetime. When an object starts playing a particular role, the properties defined for this role is added to the object, and when it stops playing a particular role, the corresponding role properties are removed. Hence, the current properties of an object is defined by the properties of the roles that it currently plays. Thus roles allow for objects to dynamically acquire new properties, and later release them.

Furthermore, as illustrated to the right in figure 4, UML application models will be defined separately to work on particular information models (i.e., roles from particular information models). The application models primarily contain classes (i.e., families of related coclasses=components) from which "stateless" application objects are instantiated to execute services made available locally on the server (to other application objects) and/or to remote clients.

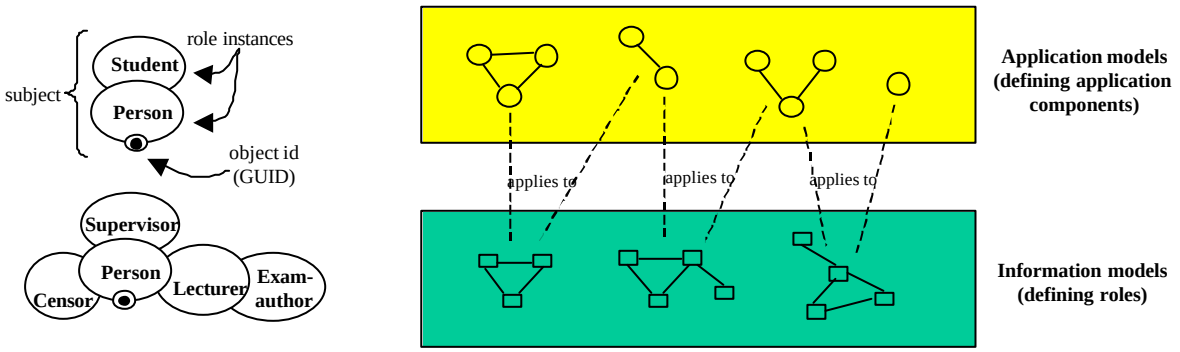

Figure 4. *The SINAI information and application organisation.*

Consequently, different sub-systems and sub-models of an overall system are defined by different UML models to avoid a single, huge and unmanageable model. For example, for electronic patient records, a separate model can be made of its "record shape and hyperlink" structure, which for Synapses EPRs correspond to nested tree-structures with local or remote hyperlinks between them. In addition there would be different models for various kinds of record content (medical documents, or parts of such documents), and yet other models for authorisation and access control, presentations (various devices), and so on.

4. Summary

SynEx has demonstrated that with state-of-the-art internet technology then it is relatively simple, at least technically, to achieve sharing and integration of distributed electronic patient records, which are themselves quite complex information entities. A number of challenges remain, however, but experience from SynEx, and other distributed information system projects [8], has lead us to believe that being "model focused" can be a great advantage for the development of web-based information systems, which essentially provide access to information from "anywhere" to "anyone"[13]. That is, UML can be useful not only for analysing and designing information systems, it can also play a key role in their implementation. The SINAI project aims to demonstrate that

- UML can be used to achieve a basic common understanding between parties involved (clients, servers, components, applications) that makes it easier to achieve integration and interoperability of information and applications.
- UML can be used to standardise the use of XML and avoid XML schema "explosion".
- Code generation from UML models can be used to achieve more standardised architectures (at least for a particular product line).
- Model and component integration techniques can be used to support development of generic and extendable information systems.

Basically, the means to achieve this are via a *"loose" integration* of distributed information by the combined use of UML and XML (figure 2), and a *"tight" integration* of applications by the combined use of role modelling techniques and component technology (figure 4).

5. References

[1] SynEx Homepage, http://www.gesi.it/synex/

[2] B.Jung, E.P.Andersen, J.Grimson; *Using XML for Seamless Integration of Distributed Electronic Patient Records*; XML Scandinavia 2000 Conference, Gothenburg, Sweden, May 2-4, 2000.

[3] B.Jung, E.P.Andersen, J.Grimson; *SynExML as a Vehicle for Electronic Patient Records*; XML Europe 2000 Conference, Paris, France, June 12-16, 2000.

[4] E.P.Andersen; *Seamless Integration of Distributed Electronic Patient Records*; Deliverable WP2 D2.1 til EU prosjektet SynEx. http://www.nr.no/~egil/ shs025-synex-client-wp2-d21.pdf

[5] E.P.Andersen; *A Platform for Electronic Patient Record Integration*; Deliverable WP2 D2.2 til EU prosjektet SynEx. http://www.nr.no/~egil/shs024-synex-server-wp2-d22.pdf

[6] Synapses Homepage, http://www.cs.tcd.ie/synapses/public/

[7] P.Hurlen, K.Skifjeld, E.P.Andersen, *The Basic Principles of the Synapses Federated Healthcare Record Server*, International Journal of Medical Informatics, Vol. 52, Nr. 1-3, 1998

[8] E.P.Andersen, Bjørn Egil Hansen; *Providing Persistent Objects to Globally Distributed Sites*; NOSA'99, 2nd Nordic Workshop on Software Architecture, University of Karlskrona/Ronneby, 12-13.August 1999, Ronneby, Sweden. http://www.nr.no/~egil/brix-ws.pdf

[9] World Wide Web Consortium; *XML*; http://www.w3.org/XML

[10] *SOAP specification*, http://msdn.microsoft.com/xml/general/SOAP_V09.asp

[11] D.Box; *A Young Person's Guide to The Simple Object Access Protocol: SOAP Increases Interoperability Across Platforms and Languages*; http://msdn.microsoft.com/msdnmag/issues/0300/soap/soap.asp

[12] Object Management Group (OMG), *CORBA*, http://www.omg.org/corba

[13] Object Management Group (OMG), *UML*, http://www.omg.org/uml

[14] Rational Rose, *UML Resource Center*, http://www.rational.com/uml

[15] E.P.Andersen; *Information Models for Component Design and Implementation*; ICSSEA'99, 12th Int'l Conf. on Software Systems Engineering and Applications December 8-10, 1999, Paris . http://www.nr.no/~egil/icssea99-im-comp.pdf , http://www.nr.no/~egil/icssea99-slides-im-comp.ppt

[16] BizTalk; http://www.biztalk.org

[17] OASIS; http://www.oasis-open.org

[18] XML ORG; http://www.xml.org

[19] Microsoft, *SQL Server*, http://www.microsoft.com/sql

[20] Microsoft, *COM/DCOM*, http://www.microsoft.com/com

[21] Microsoft, *UDA/OLE DB/ADO*, http://www.microsoft.com/data

[22] Microsoft, *IIS/ASP*, http://www.microsoft.com/iis

[23] B.B.Kristensen, *Object-Oriented Modeling with Roles*, Proc.of the 2'rd International Conference on Object-Oriented Information Systems, OOIS'95, Dublin, Ireland, 1995

[13] in a secure way - but security issues are outside the scope of this paper.