openEHR Release 1

***open*EHR Architecture**

Architecture Overview

Editors:{T Beale, S Heard}[1]

Revision: 1.1

Pages: 65

[1] Ocean Informatics Australia

email: info@openEHR.org **web**: http://www.openEHR.org

Copyright Notice

Amendment Record

Issue	Details	Raiser	Completed
RELEASE 1.0.1			
1.1	**CR-000203**: Release 1.0 explanatory text improvements. Improved path explanation. Slight re-ordering of main headings.	T Beale	08 Jul 2006
	CR-000200. Correct package names in RM diagram.	D Lloyd	
	Added configuration management and versioning material from Common IM. Added section on ontological landscape. Added section on aims. Added section on systems architectures. Added section on security.	T Beale	
RELEASE 1.0			
1.0	Initial Writing **-** content taken from Roadmap document. **CR-000147**. Make DIRECTORY Re-usable **CR-000167**. Move AOM description package to resource package in Common IM. **CR-000185**: Improved EVENT model.	T Beale	29 Jan 2006

Acknowledgements

The work reported in this paper has been funded by the University College, London; Ocean Informatics, Australia.

CORBA is a trademark of the Object Management Group

.Net is a trademark of Microsoft Corporation

1 Introduction

1.1 Purpose

This document provides an overview of the *open*EHR architecture in terms of a model overview, key global semantics, relationship to published standards, and finally the approach to building Implementation Technology Specifications (ITSs). Semantics specific to each information, archetype and service model are described in the relevant model.

The intended audience includes:

- Standards bodies producing health informatics standards
- Software development groups using *open*EHR
- Academic groups using *open*EHR
- The open source healthcare community

This document is the key technical overview of *open*EHR, and should be read before all other technical documents.

1.2 Status

This document is under development, and is published as a proposal for input to standards processes and implementation works.

This document is available at http://svn.openehr.org/specification/TAGS/Release-1.0/publishing/architecture/overview.pdf.

The latest version of this document can be found at http://svn.openehr.org/specification/TRUNK/publishing/architecture/overview.pdf.

New versions are announced on openehr-announce@openehr.org.

Blue text indicates sections under active development.

1.3 Peer review

Areas where more analysis or explanation is required are indicated with “to be continued” paragraphs like the following:

To Be Continued: more work required

Reviewers are encouraged to comment on and/or advise on these paragraphs as well as the main content. Please send requests for information to info@openEHR.org. Feedback should preferably be provided on the mailing list openehr-technical@openehr.org, or by private email.

2 Overview

This document provides an overview of the *open*EHR architecture. It commences with a description of the specification project, followed by an overview of the reference model structure and packages. Key global semantics including security, archetyping, identification, version and paths are then described. The relationship to published standards is indicated, and finally, the approach to building Implementation Technology Specifications (ITSs) is outlined.

2.1 The *open*EHR Specification Project

FIGURE 1 illustrates the *open*EHR Specification Project. The project consists of requirements, architectural specifications, implementation technology specifications (ITSs), and conformance specifications. The focus of this document is the architectural and implementation technology specifications (ITSs).

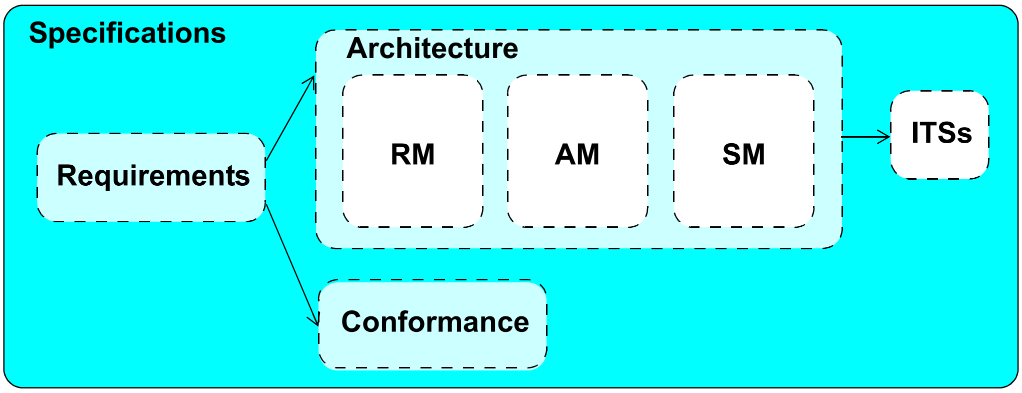


FIGURE 1 *open*EHR Specification project

The architecture specifications consist of the Reference Model (RM), the Service Model (SM) and Archetype Model (AM). The first two correspond to the ISO RM/ODP information and computational viewpoints respectively. The latter formailses the bridge between information models and knowledge resources.

The architecture specifications published by *open*EHR are defined as a set of *abstract* models, using the UML notation and formal textual class specifications. These models remain the primary references for all semantics, regardless of developments in any implementation environment. The *open*EHR Modelling Guide describes the semantics of the models. The presentation style of these abstract specifications is deliberately intended to be clear and semantically close to the ideas being communicated. Conversely, the specifications do not follow idioms or limitations of particular programming languages, schema languages or other formalisms. All such expressions are treated as ITSs, for which explicit mappings generally have to be developed and described (since almost no implementation formalism natively implements complete UML semantics).

There are numerous implementation technologies, ranging from programming languages, serial formalisms such as XML, to database and distributed object interfaces. Each of these has its own limits and strengths. The approach to implementing any of the *open*EHR abstract models in a given implementation technology is to firstly define an “implementation technology specification” (ITS) for the particular technology, then to use it to formally map the abstract models into expressions in that technology.

3 Aims of the *open*EHR Architecture

3.1 Overview

This section provides a brief overview of the requirements underpinning the *open*EHR architecture. The *open*EHR architecuture embodies 15 years of research from numerous projects and standards from around the world. It has been designed based on requirements captured over many years.

Because the architecture is highly generic, and particularly due to being archetype-driven, it satisfies many requirements outside the original concept of the "clinical EHR". For example, the same reference architecture can be used for veterinary health or even "care" of public infrastructure or listed buildings. This is due to the fact that the reference model embodies only concepts relating to "service and administrative events relating to a subject of care"; it is in archetypes and templates that specifics of the kinds of care events and subjects of care are defined. In another dimension, although one of the requirements for the *open*EHR EHR was a "patient-centric, longitudinal, shared care EHR", it is not limited to this, and can be used in purely episodic, specialist situations, e.g. as a radiology department record system. Requirements for various flavours of "health care record" can be categorised according to the two dimensions, scope, and kind of subject, as shown in FIGURE 2.

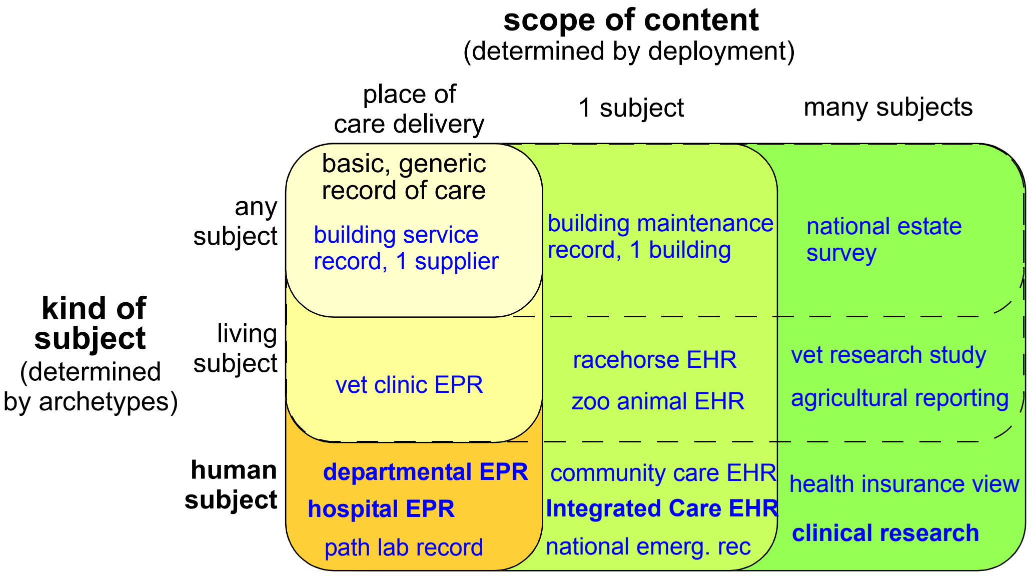


FIGURE 2 Structure of Requirements met by *open*EHR

In this figure, each bubble represents a set of requirements, being a superset of all requirements of bubbles contained within it. Requirements for a generic record of care for any kind of subject in a local deployment are represented by the top left bubble. The subsequent addition of requirements corresponding to living subjects and then human subjects is represented by the bubbles down the left side of the diagram. The requirements represented by the largest bubble on the left hand side correspond to "local health records for human care", such as radiology records, hospital EPRs and so on. Additional sets of requirements represented by wider bubbles going across the diagram correspond to extending the scope of the content of the care record first to a whole subject (resulting in a patient-centric, longitudinl health record) and then to populations or cohorts of subjects, as done in population health and research. From the (human) healthcare point of view, the important requirements groups extend all the way to the bottom row of the diagram.

Going down the diagram, requirements corresponding to increasing specificity of subject of care (from "any" to "human") are mostly implemented in *open*EHR by the use of archetypes. Going across the diagram, the requirements corresponding to increasing scope of record content (from episodic to population) are mainly expressed in different deployments, generally going from standalone to a shared interoperable form. One of the key aspirations for EHRs today is the "integrated care record" sought by many health authorities today[1], which provides an informational framework for integrated shared care.

As a result of the approach taken by *open*EHR, components and applications built to satisfy the requirements of an integrated shared care record can also be deployed as (for example) an episodic radiology record system.

Some of the key requirements developed during the evolution of GEHR to *open*EHR are listed in the following sections, corresponding to some of the major requirements groups of FIGURE 2.

Generic Care Record Requirements

The *open*EHR requirements include the following, corresponding to a basic, generic record of care:

- prioritisation of the patient / carer interaction (over e.g. research use of the record);
- suitable for all care settings (primary, acute etc);
- medico-legal faithfulness, traceability, audit-trailing;
- technology & data format independence;
- highly maintainable and flexible software;
- support for clinical data structures: lists, tables, time-series, including point and interval events.

Health Care Record (EPR)

The following requirements addressed in *open*EHR correspond to a local health record, or EPR:

- support for all aspects of pathology data, including normal ranges, alternative systems of units etc;
- supports all natural languages, as well as translations between languages in the record;
- integrates with any/multiple terminologies.

Shared Care EHR

The following requirements addressed in *open*EHR corespond to an integrated shared care EHR:

- support for patient privacy, including anonymous EHRs;
- facilitate sharing of EHRs via interoperability at data and knowledge levels;
- compatibility with CEN 13606, Corbamed, and messaging systems;
- support semi-automated and automated distributed workflows.

3.2 Clinical Aims

From a more specifically clinical care perspective (rather than a record-keeping perspective), the following requirements have been identified during the development of *open*EHR:

[1] The Integrated Care EHR is described in ISO/TR 20514 (2005)

- The need for a patient-centric, lifelong electronic health record that entails a holistic view of patient needs as opposed to niche problem-solving and decision-support techniques for limited diagnostic purposes;
- Integration of different views of the patient (GP, emergency and acute care, pathology, radiology, computerised patient-order entry, etc.) with the vast body of available knowledge resources (terminologies, clinical guidelines and computerised libraries);
- Clinical decision-support to improve patient safety and reduced costs through repeated medical investigations;
- Access to standards-based computing applications.

The Integrated Care EHR holds great promise: to generalise and make widely available the benefits of computerisation that have been demonstrated individually and in isolated settings. These can be summarised as:

- Reducing adverse events arising from medication errors such as interactions, duplications or inappropriate treatments and the flow-on costs associated with these;
- Improving the timely access to critical information and reduced clinician time searching for information;
- Reducing the incidence of patients being overlooked in the healthcare system due to information not being communicated;
- Reducing the duplication of investigations and other tests and procedures due to results not being available in the local computing environment;
- Improved prevention and early detection, based on predictive risk factor analysis, which is possible with quality EHR data;
- Improved decision making through decision support tools with access to the patient's whole EHR;
- Improving access to and computation of evidence based guidelines;
- Increasing targeted health initiatives known to be effective, based on patient criteria; and
- Reduced hospitalisations and readmissions.

One comprehensive statement of EHR requirements covering many of the above is the ISO Technical Report 18308[1] for which an *open*EHR profile has been created[2]. The requirements summarised above are described in more detail in the *open*EHR EHR Information Model document.

3.3 Deployment Environments

Ultimately any software and information architecture only provides utility when deployed. The architecture of *open*EHR is designed to support the construction of a number of types of system. One of the most important, the integrated shared care health record is illustrated in FIGURE 3.

In this form, the *open*EHR services are added to the existing IT infrastructure to provide a shared, secure health record for patients that are seen by any number of health providers in their community context. *open*EHR-enabled systems can also be used to provide EMR/EPR functionality at provider locations. Overall, a number of important categories of system can be implemented using *open*EHR including the following:

[1] see http://www.openehr.org/downloads/ISOEHRRequirements.zip

[2] see http://svn.openehr.org/specification/TRUNK/publishing/requirements/iso18308_conformance.pdf

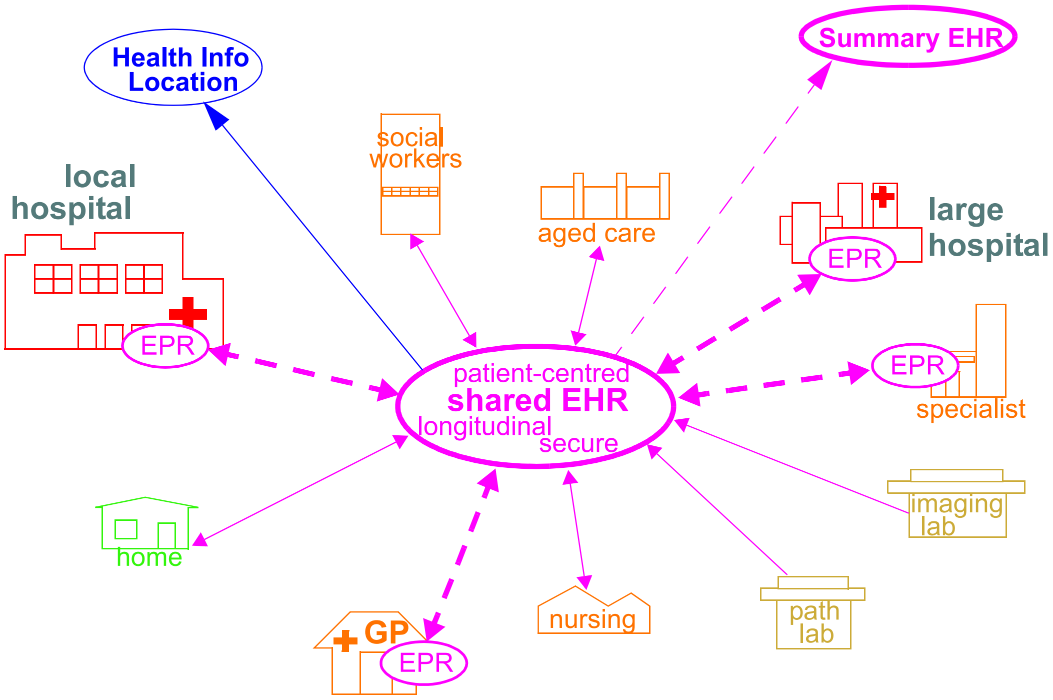


FIGURE 3 Community Shared-care Context

- shared-care community or regional health service EHRs;
- summary EHRs at a national, state, province or similar level;
- small desktop GP systems;
- hospital EMRs;
- consolidated and summary EHRs in federation environments;
- legacy data purification and validation gateways;
- web-based secure EHR systems for mobile patients.

4 Design Principles

The *open*EHR approach to modelling information, services and domain knowledge is based on a number of design principles, described below. The application of these principles lead to a separation of the models of the *open*EHR architecture, and consequently, a high level of componentisation. This leads to better maintainability, extensibility, and flexible deployment.

4.1 Ontological Separation

The most basic kind of distinction in any system of models is ontological, i.e. in the levels of abstraction of description of the real world. All models carry some kind of semantic content, but not all semantics are the same, or even of the same category. For example, some part of the SNOMED-CT[1] terminology describes types of bacterial infection, sites in the body, and symptoms. An *information* model might specify a logical type Quantity. A *content* model might define the model of information collected in an ante-natal examination by a physician. These types of “information” are qualitatively different, and need to be developed and maintained separately within the overall model eco-system. FIGURE 4 illustrates these distinctions, and indicates what parts are built directly into software and databases.

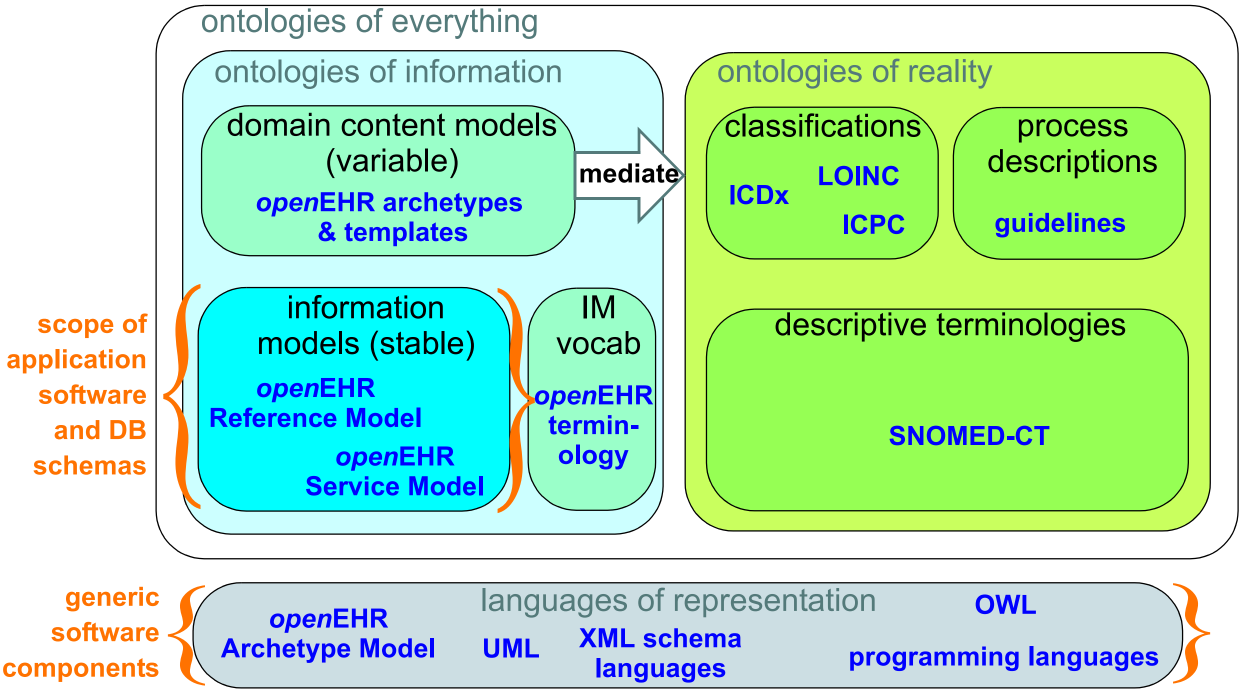


FIGURE 4 The Ontological Landscape

This figure shows a primary separation between “ontologies of information” i.e. models of information content, and “ontologies of reality” i.e. descriptions and classifications of real phenomena. These two categories have to be separated because the type of authors, the representation and the purposes are completely different. In health informatics, this separation already exists by and large, due to the development of terminologies and classifications.

[1] See http://www.snomed.org

A secondary ontological separation within the information side is shown between information models and domain content models. The former category corresponds to semantics that are *invariant* across the domain (e.g. basic data types like coded terms, data structures like lists, identifiers), wwhile the latter corresponds to *variable* domain level content descriptions - descriptions of information structures such as “microbiology result” rather than descriptions of actual phenomena in the real world (such as infection by a microbe). This separation is not generally well understood, and historically, a great deal of domain-level semantics has been hard-wired into the software and databases, leading to relatively unmaintainable systems.

By clearly separating the three categories - information models, domain content models, and terminologies - the *open*EHR architecture enables each to have a well-defined, limited scope and clear interfaces. This limits the dependence of each on the other, leading to more maintainable and adaptable systems.

Two-level Modelling and Archetypes

One of the key paradigms on which *open*EHR is based is known as “two-level” modelling, described in [2]. Under the two-level approach, a stable reference information model constitutes the first level of modelling, while formal definitions of clinical content in the form of archetypes and templates constitute the second. *Only the first level* (the Reference Model) *is implemented in software*, significantly reducing the dependency of deployed systems and data on variable content definitions. The only other parts of the model universe implemented in software are highly stable languages/models of representation (shown at the bottom of FIGURE 4). As a consequence, systems have the possibility of being far smaller and more maintainable than single-level systems. They are also inherently self-adapting, since they are built to consume archetypes and templates as they are developed into the future.

Archetypes and templates also act as a well-defined semantic gateway to terminologies, classifications and computerised clinical guidelines. The alternative in the past has been to try to make systems function solely with a combination of hard-wired software and terminology. This approach is flawed, since terminologies don’t contain definitions of domain content (e.g. “microbiology result”), but rather facts about the real world (e.g. kinds of microbes and the effects of infection in humans).

The use of archetyping in *open*EHR engenders new relationships between information and models, as shown in FIGURE 5.

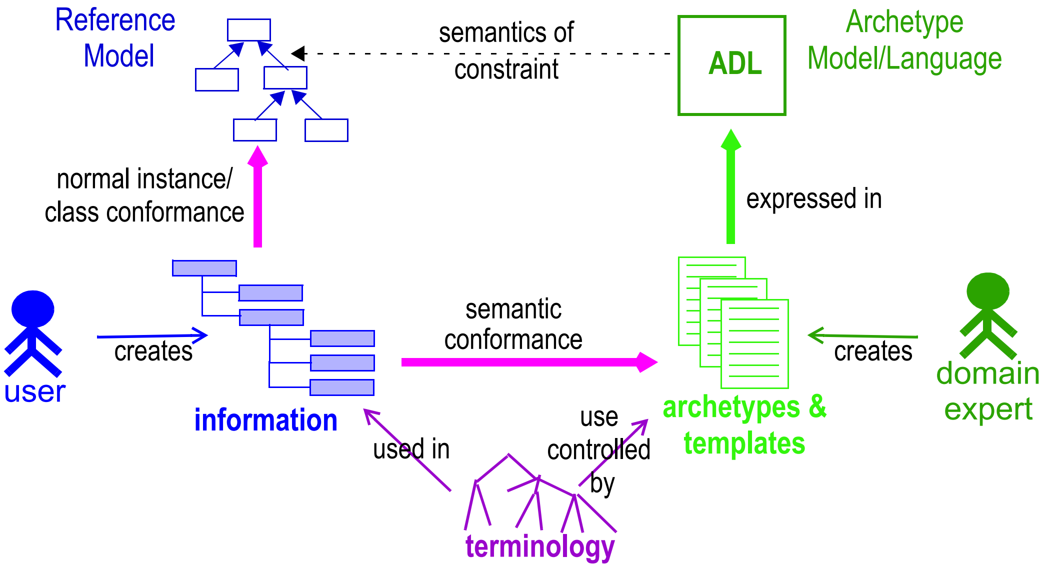


FIGURE 5 Archetype Meta-architecture

In this figure, “data” as we know it in normal information systems (shown on the bottom left) conforms in the usual way to an object model (top left). Systems engineered in the “classic” way (i.e. all domain semantics are encoded somewhere in the software or database) are limited to this kind of architecture. With the use of two-level modelling, runtime data now conform semantically to archetypes as well as concretely to the reference model. All archetypes are expressed in a generic Archetype Definition Language (ADL).

The details of how archetypes and templates work in *open*EHR are described in Archetyping on page 48.

Consequences for Software Engineering

Two-level modelling significantly changes the dynamics of the systems development process. In the usual IT-intensive process, requirements are gathered via *ad hoc* discussions with users (typically via the well-known “use case” methodology), designs and models built from the requirements, implementation proceeds from the design, followed by testing and deployment and ultimately the maintenance part of the lifecycle. This is usually characterised by ongoing high costs of implementation change and/or a widening gap between system capabilities and the requirements at any moment. The approach also suffers from the fact that *ad hoc* conversations with systems users nearly always fails to reveal underlying content and workflow. Under the two-level paradigm, the core part of the system is based on the reference and archetype models (includes generic logic for storage, querying, caching etc), both of which are extremely stable, while domain semantics are mostly delegated to domain specalists who work building archetypes (reusable), templates (local use) and terminology (general use). The process is illustrated in FIGURE 6. Within this process, IT developers concentrate on generic components such as data management and interoperability, while groups of domain experts work outside the software development process, generating definitions that are used by systems at runtime.

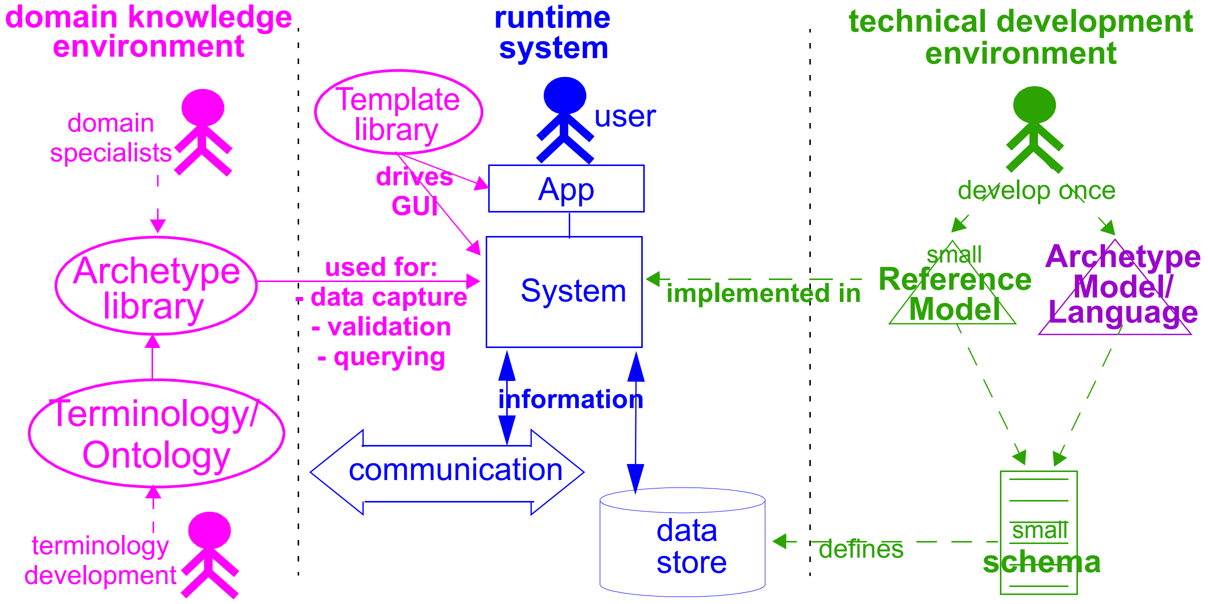


FIGURE 6 Two-level Software Engineering

Clearly applications cannot always be totally generic (although many data capture and viewing applications are); decision support, administrative, scheduling and many other applications still require custom engineering. However, all such applications can now rely on an archetype- and template-driven computing platform. A key result of this approach is that archetypes now constitute a technology-independent, single-source expression of domain semantics, used to drive database schemas, software logic, GUI screen definitions, message schemas and all other technical expressions of the semantics.

4.2 Separation of Responsibilities

A second key design paradigm used in *open*EHR is that of separation of responsibilities within the computing environment. Complex domains are only tractable if the functionality is first partitioned into broad areas of interest, i.e. into a “system of systems” [4]. This principle has been understood in computer science for a long time under the rubrics “low coupling”, “encapsulation” and “componentisation”, and has resulted in highly successful frameworks and standards, including the OMG’s CORBA specifications and the explosion of object-oriented languages, libraries and frameworks. Each area of functionality forms a focal point for a set of models formally describing that area, which, taken together usually correspond to a distinct information system or service.

FIGURE 7 illustrates a notional health information environment containing numerous services, each denoted by a bubble. Typical connections are indicated by lines, and bubbles closer to the centre correspond to services closer to the core needs of clinical care delivery, such as the EHR, terminology, demographics/identification and medical reference data. Of the services shown on the diagram, *open*EHR currently provides specifications only for the more central ones, including EHR and Demographics.

Since there are standards available for some aspects of many services, such as terminology, image formats, messages, EHR Extracts, service-based interoperation, and numerous standards for details such as date/time formats and string encoding, the *open*EHR specifications often act as a mechanism to integrate existing standards.

4.3 Separation of Viewpoints

The third computing paradigm used in *open*EHR is a natural consequence of the separation of responsibilities, namely the *separation of viewpoints*. When responsibilities are divided up among distinct components, it becomes necessary to define a) the information that each processes, and b) how they will communicate. These two aspects of models constitute the two central “viewpoints” of the ISO RM/ODP model [3], marked in bold in the following:

Enterprise: concerned with the business activities, i.e. purpose, scope and policies of the specified system.

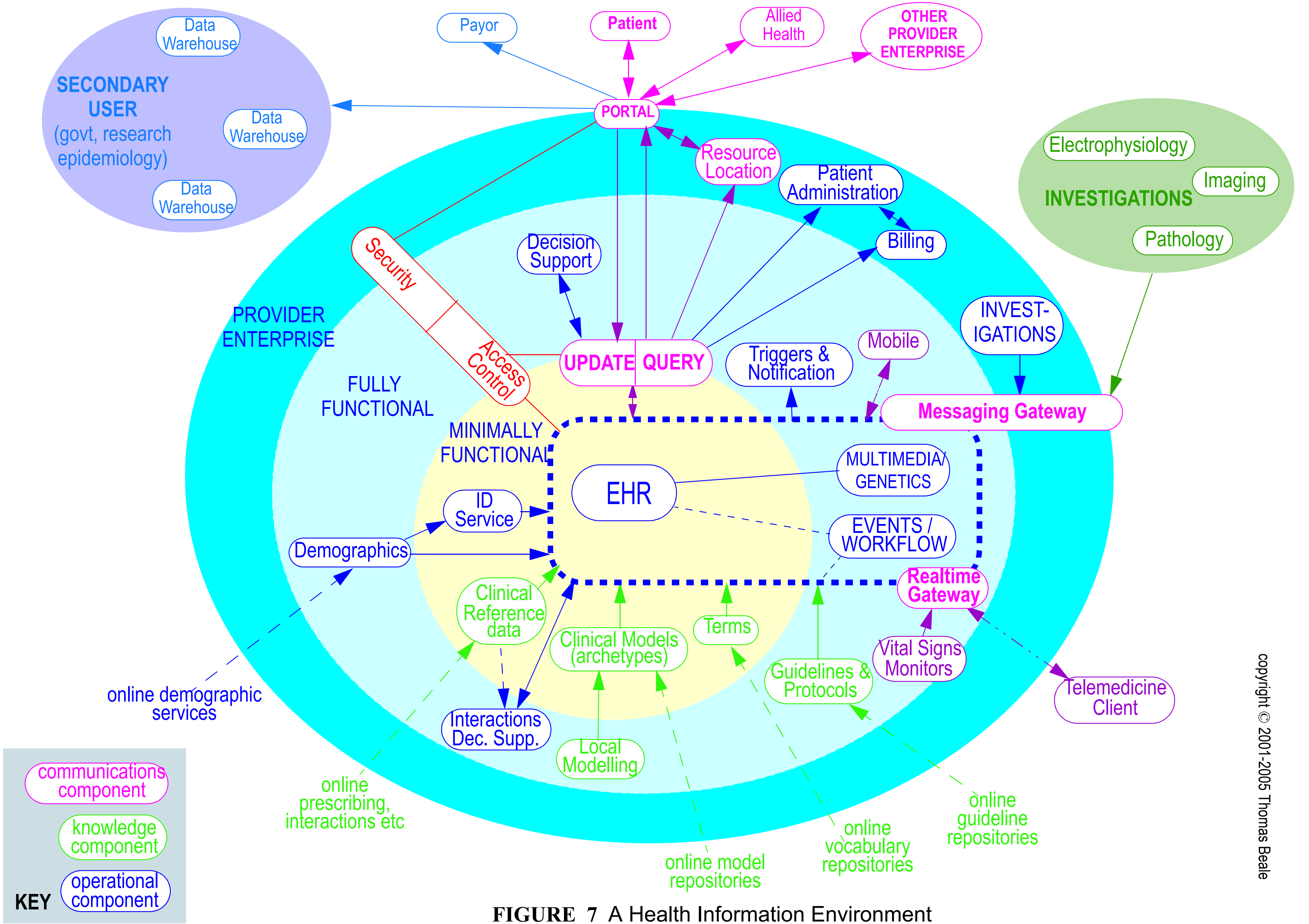
Information: concerned with the semantics of information that needs to be stored and processed in the system.

Computational: concerned with the description of the system as a set of objects that interact at interfaces - enabling system distribution.

Engineering: concerned with the mechanisms supporting system distribution.

Technological: concerned with the detail of the components from which the distributed system is constructed.

The *open*EHR specifications accordingly include an information viewpoint - the *open*EHR Reference Model - and a computational viewpoint - the *open*EHR Service Model. The Engineering viewpoint corresponds to the Implementation Technology Specification (ITS) models of *open*EHR (see Implementation Technology Specifications on page 62), while the Technological viewpoint corresponds to the technologies and components used in an actual deployment. An important aspect of the division into viewpoints is that there is generally not a 1:1 relationship between model specifications in each



copyright © 2001-2005 Thomas Beale

FIGURE 7 A Health Information Environment

viewpoint. For example, there might be a concept of “health mandate” (a CEN ENV13940 Continuity of Care concept) in the enterprise viewpoint. In the information viewpoint, this might have become a model containing many classes. In the computational viewpoint, the information structures defined in the information viewpoint are likely to recur in multiple services, and there may or may not be a “health mandate” service. The granularity of services defined in the computational viewpoint corresponds most strongly to divisions of function in an enterprise or region, while the granularity of components in the information view points corresponds to the granularity of mental concepts in the problem space, the latter almost always being more fine-grained.

5 *open*EHR Package Structure

5.1 Overview

FIGURE 10 illustrates the overall package structure of the *open*EHR formal specifications. Three major packages are defined: `rm`, `am` and `sm`. All packages defining detailed models appear inside one of these outer packages, which may also be thought of as namespaces. They are conceptually defined within the `org.openehr` namespace, which can be represented in UML as further packages. In some implementation technologies (e.g. Java), the `org.openehr` namespace may actually be used within program texts.

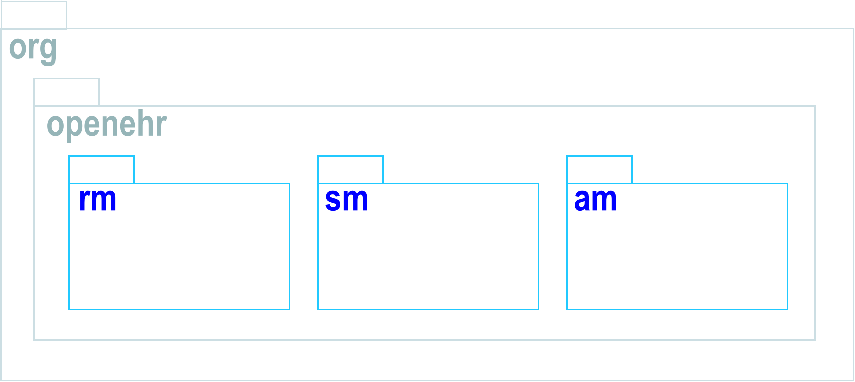


FIGURE 8 Global Package Structure of *open*EHR

One of the important design aims of *open*EHR is to provide a coherent, consistent and re-usable type system for scientific and health computing. Accordingly, a "common type platform" is defined in the lower part of the RM, providing identifiers, data types, data structures and various common design patterns that can be re-used ubiquitously in the upper layers of the RM, and equally in the AM and SM packages. FIGURE 9 illustrates the relationships between the outer packages and their main constituents. The common type platform is shown at the bottom.

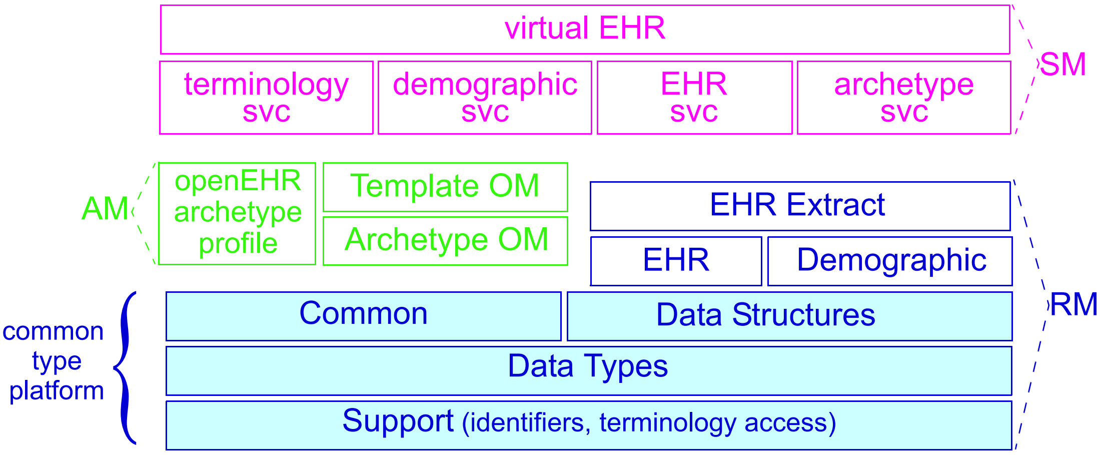


FIGURE 9 Computing Platform view of packages

5.2 Reference Model (RM)

Ech package defines a local context for definition of classes. FIGURE 10 illustrates the RM package structure. An informal division into “scientific computing” and “health information “ is shown. The packages in the latter group are generic, and are used by all *open*EHR models, in all the outer packages. Together, they provide identification, access to knowledge resources, data types and structures, versioning semantics, and support for archetyping. The packages in the former group define the semantics of enterprise level health information types, including the EHR and demographics.

Each outer package in FIGURE 10 corresponds to one *open*EHR specification document[1], documenting an “information model” (IM). The package structure will normally be replicated in all ITS expressions, e.g. XML schema, programming languages like Java, C# and Eiffel, and interoperability definitions like WSDL, IDL and .Net.

5.2.1 Package Overview

The following sub-sections provide a brief overview of the RM packages.

Support Information Model

This package describes the most basic concepts, required by all other packages, and is comprised of the Definitions, Identification, Terminology and Measurement packages. The semantics defined in these packages allow all other models to use identifiers and to have access to knowledge services like terminology and other reference data. The support package includes the special package `assumed_types`, describing what basic types are assumed by *open*EHR in external type systems; this package is a guide for integrating *open*EHR models proper into the type systems of implementation technologies.

Data Types Information Model

A set of clearly defined data types underlies all other models, and provides a number of general and clinically specific types required for all kinds of health information. The following categories of datatypes are defined in the data types reference model.

Text: plain text, coded text, paragraphs.

Quantities: any ordered type including ordinal values (used for representing symbolic ordered values such as “+”, “++”, “+++”), measured quantities with values and units, and so on.

Date/times: date, time, date-time types, and partial date/time types.

Encapsulated data: multimedia, parsable content.

Basic types: boolean, state variable.

Data Structures Information Model

In most *open*EHR information models, generic data structures are used for expressing content whose particular structure will be defined by archetypes. The generic structures are as follows.

Single: single items, used to contain any single value, such as a height or weight.

List: linear lists of named items, such as many pathology test results.

Table: tabular data, including unlimited and limited length tables with named and ordered columns, and potentially named rows.

Tree: tree-shaped data, which may be conceptually a list of lists, or other deep structure.

[1] with the exception of the EHR and Composition packages, which are both described in the EHR Reference Model document.

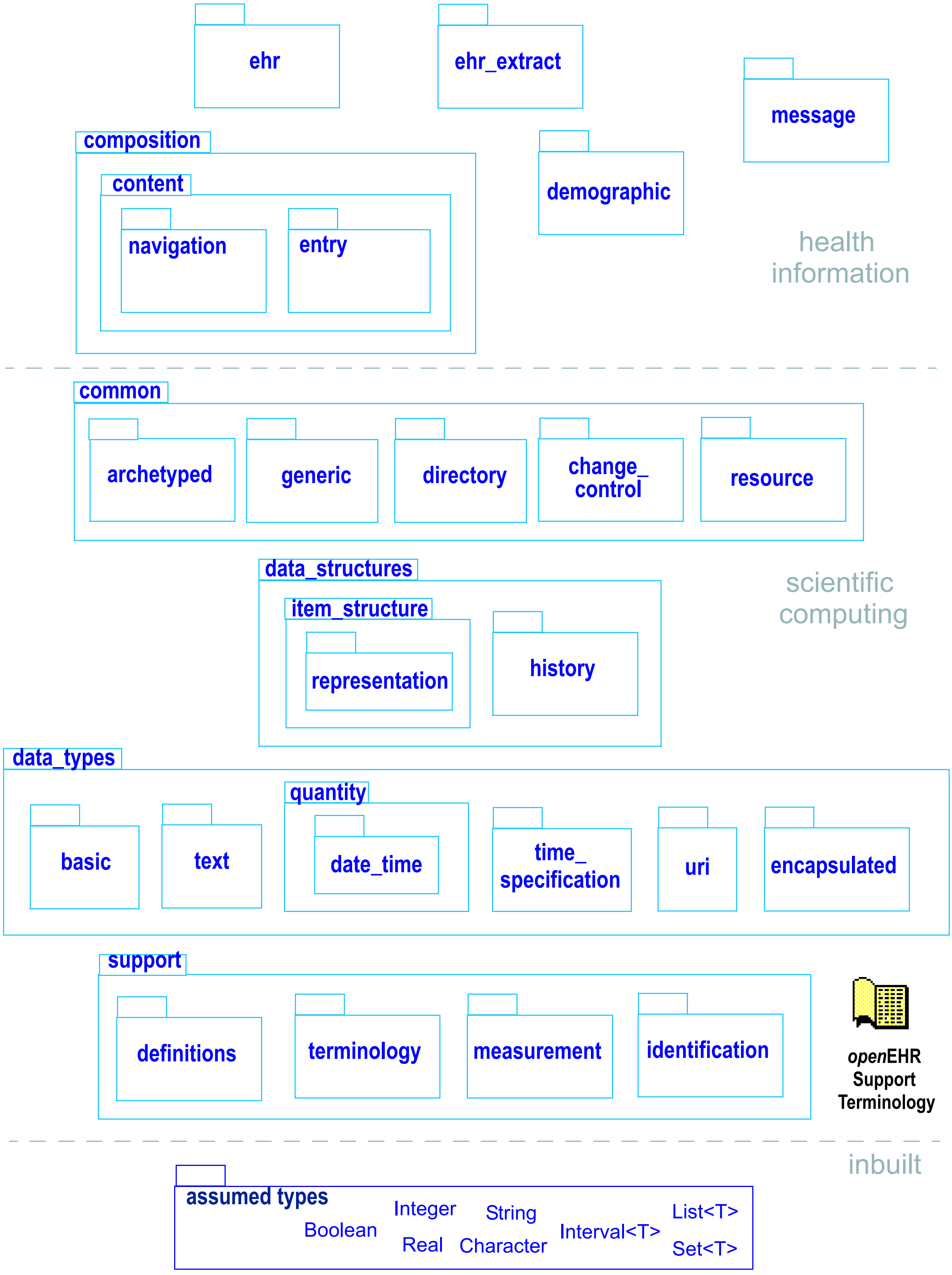


FIGURE 10 Structure of org.openehr.rm package

History: time-series structures, where each time-point can be an entire data structure of any complexity, described by one of the above structure types. Point and interval samples are supported.

Common Information Model

Several concepts recur in higher level packages. The classes `LOCATABLE` and `ARCHETYPED` provide the link between information and archetype models. The classes `ATTESTATION` and `PARTICIPATION` are generic domain concepts that appear in various reference models. The `change_control` package defines a formal model of change management and versioning which applies to any service that needs to be able to supply previous states of its information, in particular the demographic and EHR services. The key semantics of versioning in *open*EHR are described in section 8 on page 41.

EHR Information Model

The EHR IM defines the containment and context semantics of the concepts `EHR`, `COMPOSITION`, `SECTION`, and `ENTRY`. These classes are the major coarse-grained components of the EHR, and correspond directly to the classes of the same names in CEN EN13606:2005 and fairly closely to the "levels" of the same names in the HL7 Clinical Document Architecture (CDA) release 2.0.

EHR Extract

The EHR Extract IM defines how an EHR extract is built from `COMPOSITIONs`, demographic, and access control information from the EHR. A number of Extract variations are supported, including "full *open*EHR", a simplified form for integration with CEN EN13606, and an *open*EHR/*open*EHR synchronisation Extract.

Demographics

The demographic model defines generic concepts of `PARTY`, `ROLE` and related details such as contact addresses. The archetype model defines the semantics of constraint on `PARTYs`, allowing archetypes for any type of person, organisation, role and role relationship to be described. This approach provides a flexible way of including the arbitrary demographic attributes allowed in the OMG HDTF PIDS standard.

Workflow

Workflow is the dynamic side of clinical care, and consists of models to describe the semantics of processes, such as recalls, as well as any care process resulting from execution of guidelines.

5.3 Archetype Model (AM)

The *open*EHR `am` package contains the models necessary to describe the semantics of archetypes and templates, and their use within *open*EHR. These include ADL, the Archetype Definition Language (expressed in the form of a syntax specification), the `archetype` and `template` packages, defining the object-oriented semantics of archetypes and templates, and the `openehr_profile` package, which defines a profile of the generic archetype model defined in the `archetype` package, for use in *open*EHR (and other health computing endeavours). The internal structure of the `am` package is shown in FIGURE 11.

5.4 Service Model (SM)

The *open*EHR service model includes definitions of basic services in the health information environment, centred around the EHR. It is illustrated in FIGURE 12. The set of services actually included will undoubtedly evolve over time, so this diagram should not be seen as definitive.

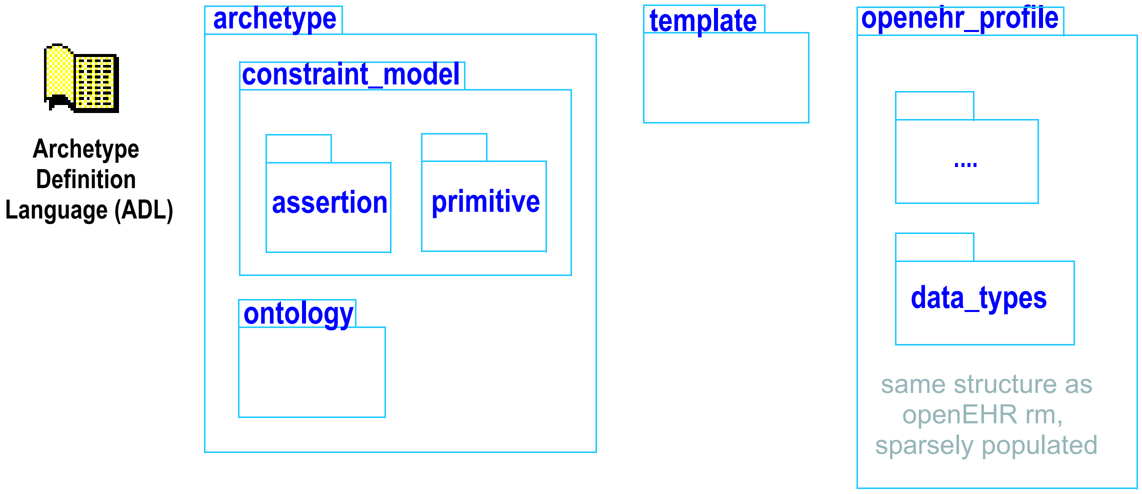


FIGURE 11 Structure of the org.openehr.am package

Virtual EHR API

The virtual EHR API defines the fine-grained interface to EHR data, at the level of Compositions and below. It allows an application to create new EHR information, and to request parts of an existing EHR and modify them. This API enables fine-grained archetype-mediated data manipulation. Changes to the EHR are committed via the EHR service.

EHR Service Model

The EHR service model defines the coarse-grained interface to electronic health record service. The level of granularity is *open*EHR Contributions and Compositions, i.e. a version-control / change-set interface.

Part of the model defines the semantics of server-side querying, i.e. queries which cause large amounts of data to be processed, generally returning small aggregated answers, such as averages, or sets of ids of patients matching a particular criterion.

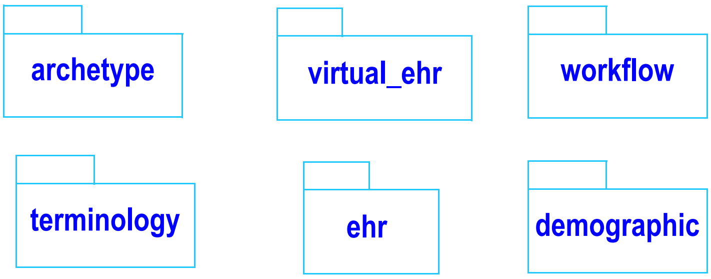


FIGURE 12 Structure of the org.openehr.sm package

Archetype Service Model

The archetype service model defines the interface to online repositories of archetypes, and can be used both by GUI applications designed for human browsing as well as access by other software services such as the EHR.

Terminology Interface Model

The terminology interface service provides the means for all other services to access any terminology available in the health information environment, including basic classification vocabularies such as

ICDx and ICPC, as well as more advanced ontology-based terminologies. Following the concept of division of responsibilites in a system-of-systems context, the terminology interface abstracts the different underlying architectures of each terminology, allowing other services in the environment to access terms in a standard way. The terminology service is thus the gateway to all ontology- and terminology-based knowledge services in the environment, which along with services for accessing guidelines, drug data and other "reference data" enables inferencing and decision support to be carried out in the environment.

6 Design of the *open*EHR EHR

6.1 The EHR System

In informational terms, a minimal EHR system based on *open*EHR consists of an EHR repository, an archetype repository, terminology (if available), and demographic/identity information, as shown in FIGURE 13.

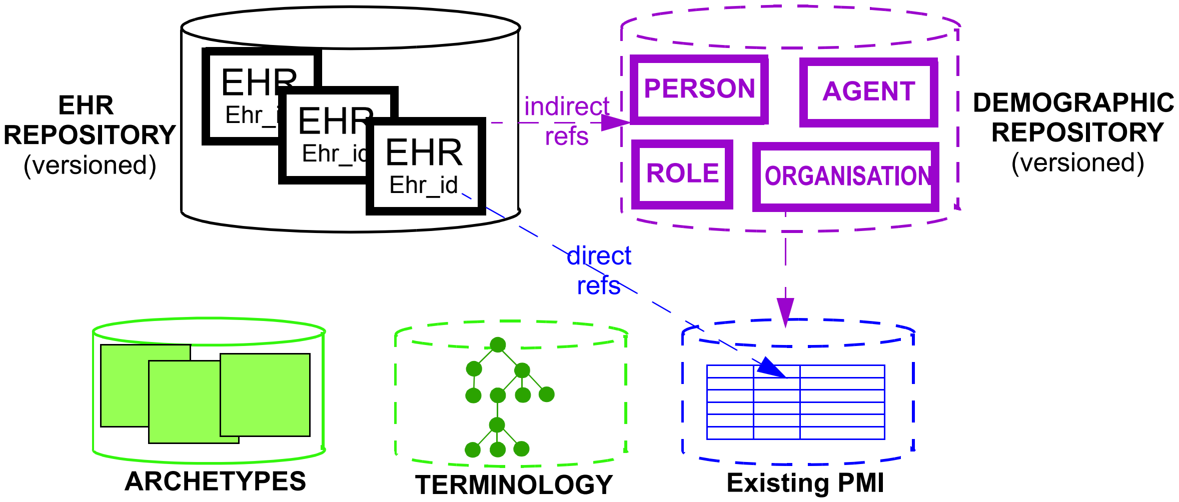


FIGURE 13 Minimal *open*EHR EHR System

The latter may be in the form of an existing PMI (patient master index) or other directory, or it may be in the form of an *open*EHR demographic repository. An *open*EHR demographic repository can act as a front end to an existing PMI or in its own right. Either way it performs two functions: standardisation of demographic information structures and versioning. EHRs contain references to entities in whichever demographic repository is available, and can be configured to include no demographic or identifying data themselves. One of the basic principles of *open*EHR is the complete separation of EHR and demographic information, such that an EHR taken in isolation contains little or no clue as to the identity of the patient it belongs to. The security benefits are described below. In more complete EHR systems, numerous other services (particularly security-related) would normally be deployed, as shown in FIGURE 7.

6.2 Top-level Information Structures

As has been shown, the *open*EHR information models define information at varying levels of granularity. Fine-grained structures defined in the Support and Data types are used in the Data Structures and Common models; these are used in turn in the EHR, EHR Extract, Demographic and other "top-level" models. These latter models define the "top-level structures" of *open*EHR, i.e. content structures that can sensibly stand alone, and may be considered the equivalent of separate documents in a document-oriented system. In *open*EHR information systems, it is generally the top-level structures that are of direct interest to users. The major top-level structures include the following:

- Composition - the committal unit of the EHR (see type `COMPOSITION` in EHR IM);
- EHR Status - the status summary of the EHR (see type `EHR_STATUS` in EHR IM);

- Folder hierarchy - act as directory structures in EHR, Demographic services (see type `FOLDER` in Common IM);
- Party - various subtypes including Actor, Role, etc representing a demographic entity with identity and contact details (see type `PARTY` and subtypes in Demographic IM);
- EHR Extract - the transmission unit between EHR systems, containing a serialisation of EHR, demographic and other content (see type `EHR_EXTRACT` in EHR Extract IM).

All persistent *open*EHR EHR, demographic and related content is found within top-level information structures. Most of these are visible in the following figures.

6.3 The EHR

The *open*EHR EHR is structured according to a relatively simple hierarchical model whose levels are: EHR, Folder, Composition, Section, Entry, and various low-level data structures and data types. These levels correspond closely to the levels in internationally agreed models of health information such as the CEN EN13606 and HL7 CDA standards. The high-level structure of the *open*EHR EHR is shown in FIGURE 14. In this figure, the parts of the EHR are as follows:

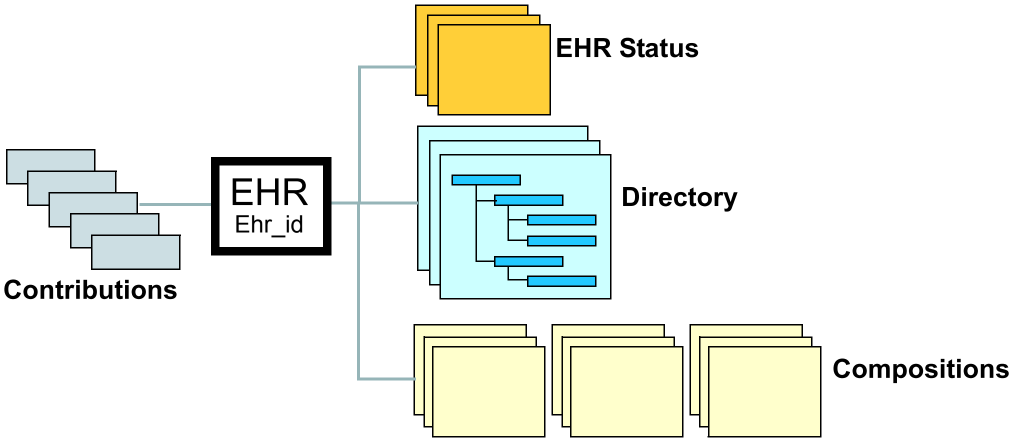


FIGURE 14 High-level Structure of the *open*EHR EHR

- *EHR*: the root object, identified by a globally unique EHR identifier;
- *EHR_status (versioned)*: an object containing various status and control information, optionally including the identifier of the subject (i.e. patient) currently associated with the record;
- *Directory (versioned)*: an optional hierarchical structure of Folders that can be used to logically organise Compositions;
- *Compositions (versioned)*: the containers of all clinical and administrative content of the record;
- *Contributions*: the change-set records for every change made to the health record; each Contribution references a set of one or more Versions of any of the versioned items in the record that were committed together by a user to an EHR system.

The logical structure of a typical Composition is shown in more detail in FIGURE 15. In this figure, the various hierarchical levels from Composition to the data types are shown in a typical arrangement. The 21 data types provide for all types of data needed for clinical and administrative recording.

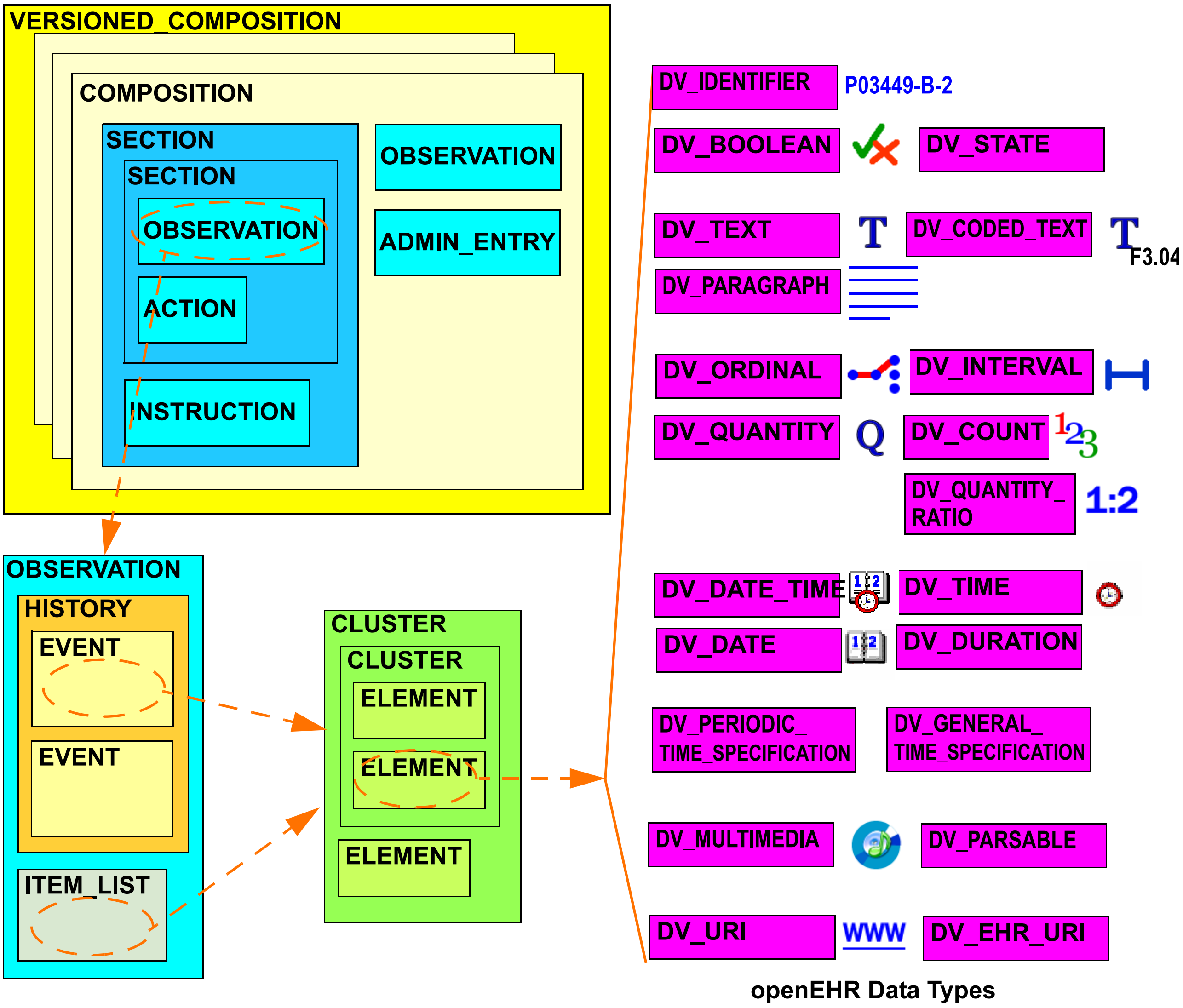


FIGURE 15 Elements of an *open*EHR Composition

6.4 Entries and "clinical statements"

All clinical information created in the *open*EHR EHR is ultimately expressed in "Entries". An Entry is logically a single 'clinical statement', and may be a single short narrative phrase, but may also contain a significant amount of data, e.g. an entire microbiology result, a psychiatric examination note, a complex medication order. In terms of actual content, the Entry classes are the most important in the *open*EHR EHR Information Model, since they define the semantics of all the 'hard' information in the record. They are intended to be archetyped, and in fact, archetypes for Entries and sub-parts of Entries make up the vast majority of archetypes defined for the EHR.

The design of Entries is based on past experience and upon a solid theoretical model of clinical recording, which includes an ontology of information categories shown in the FIGURE 16 (the *open*EHR Entry class names are annotated next to categories which are directly modelled in the Reference Model; non-italic names correspond to directly instantiable classes).

The key top-level categories in the ontology are 'care information' and 'administrative information'. The former encompasses all statements that might be recorded at any point during the care process,

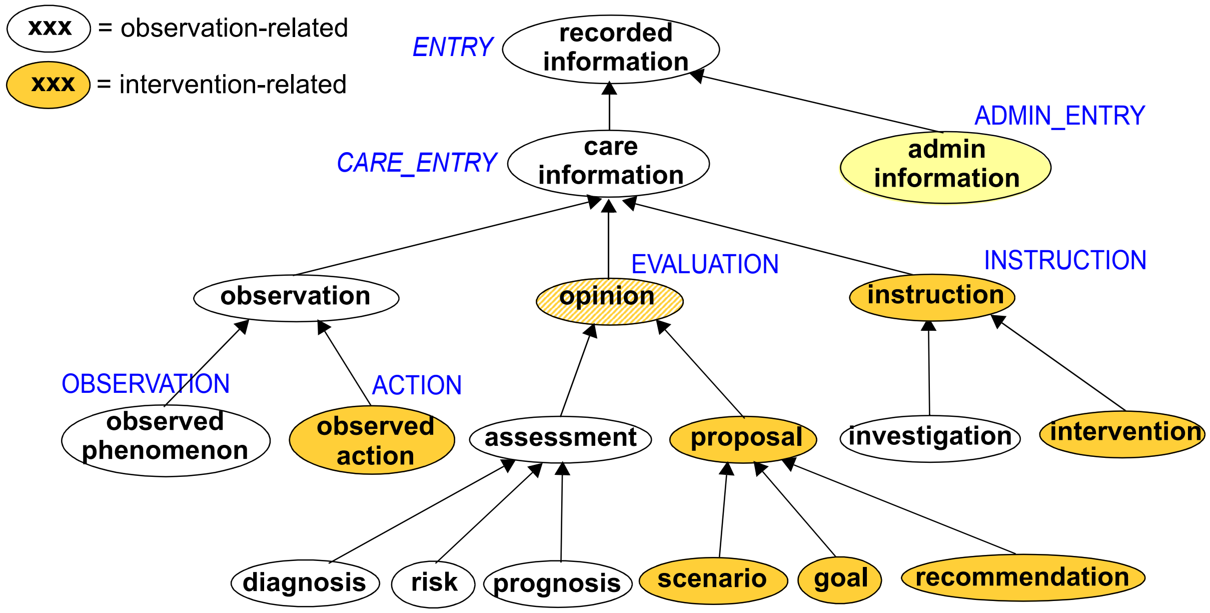


FIGURE 16 Ontology of Recorded Information

and consists of the major subcategories ‘observation’, ‘opinion’, ‘instruction’, and ‘action’ (a kind of observation) which themselves correspond to the past, present and future in time. The administrative information category covers information which is not generated by the care process proper, but relates to organising it, such as appointments and admissions. This information is not about care, but about the logistics of care delivery.

Each of the principal information categories of FIGURE 16 occurs somewhere in the cognitive process during care delivery, shown in FIGURE 17. This figure shows the cycle of information creation in an iterative, problem solving process typical not just of clinical medicine but of science itself. The system as a whole is made up of two parts: the “patient system” and the “clinical investigator system”. The latter consists of health carers, and may include the patient (at points in time when the patient performs observational or therapeutic activities), and is responsible for understanding the state of the patient system and delivering care to it.

The process illustrated in FIGURE 17 is related to Lawrence Weed’s “problem-oriented” approach of EHR recording (and its many later derivatives) and the “hypothetico-deductive model” of reasoning. However hypothesis-making and testing is not the only successful process used by clinical professionals - evidence shows that many (particularly those older and more experienced) rely on pattern recognition and direct retrieval of plans used previously with similar patients or prototype models. The investigator process is compatible with both cognitive approaches, since it does not say how opinions are formed, nor imply any specific number or size of iterations to bring the process to a conclusion. As such, the *open*EHR information model does not impose any process model, only the types of information used.

Correct use of the categories from the ontology is facilitated by using archetypes designed to map particular kinds of clinical statement to particular Entry subtypes. In a system where Entries are thus modelled, there will be no danger of incorrectly identifying the various kinds of Entries, as long as the Entry subtype, time, and certainty/negation are taken into account. Note that even if the ontology of FIGURE 16 is not correct (undoubtedly it isn’t), archetypes will be constructed to account for each improved idea of what such categories should really be.

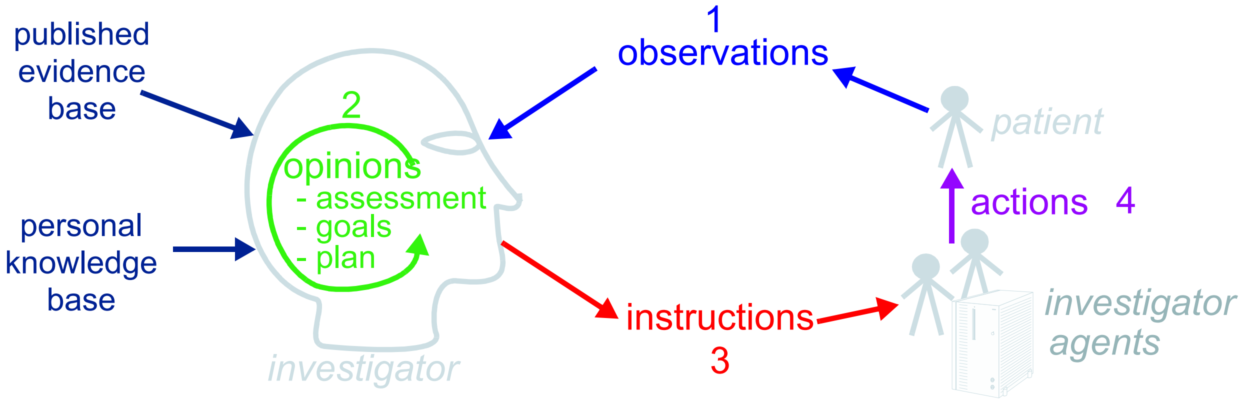


FIGURE 17 Relationship of information types to the investigation process

Clinical Statement Status and Negation

A well-known problem in clinical information recording is the assigment of "status" to recorded items. Kinds of status include variants like "actual value of P" (P stands for some phenomenon), "family history of P", "risk of P", "fear of P", as well as negation of any of these, i.e. "not/no P", "no history of P" etc. A proper analysis of these so called statuses shows that they are not "statuses" at all, but different categories of information as per the ontology of FIGURE 17. In general, negations are handled by using "exclusion" archetypes for the appropriate Entry type. For example, "no allergies" can be modelled using an Evaluation archetype that describes which allergies are excluded for this patient. Another set of statement types that can be confused in systems that do not properly model information categories concern interventions, e.g. "hip replacement (5 years ago)", "hip replacement (recommended)", "hip replacement (ordered for next tuesday 10 am)".

All of these statement types map directly to one of the *open*EHR Entry types in an unambiguous fashion, ensuring that querying of the EHR does not match incorrect data, such as a statement about fear or risk, when the query was for an observation of the phenomenon in question.

Further details on the *open*EHR model clinical information are given in the EHR IM document, Entry Section.

6.5 Managing Interventions

A key part of the investigation process shown in FIGURE 17, and indeed healthcare in general, is intervention. Specifying and managing interventions (whether the simplest prescriptions or complex surgery and therapy) is a hard problem for information systems because it is in "future time" (meaning that intervention activities have to be expressed using branching/looping time specifications, not the simple linear time of observations), unexpected events can change things (e.g. patient reaction to drugs), and the status of an given intervention can be hard to track, particularly in disributed systems. However, from the health professional's point of view, almost nothing is more basic than wanting to find out: what medications is this patient on, since when, and what is the progress?

The *open*EHR approach to these challenges is to use the Entry type Instruction, its subpart Activity to specify interventions in the future, and the Entry subtype Action to record what has actually happened. A number of important features are provided in this model, including:

- a single, flexible way of modelling all interventions, whether they be single drug medication orders or complex hospital-based therapies;

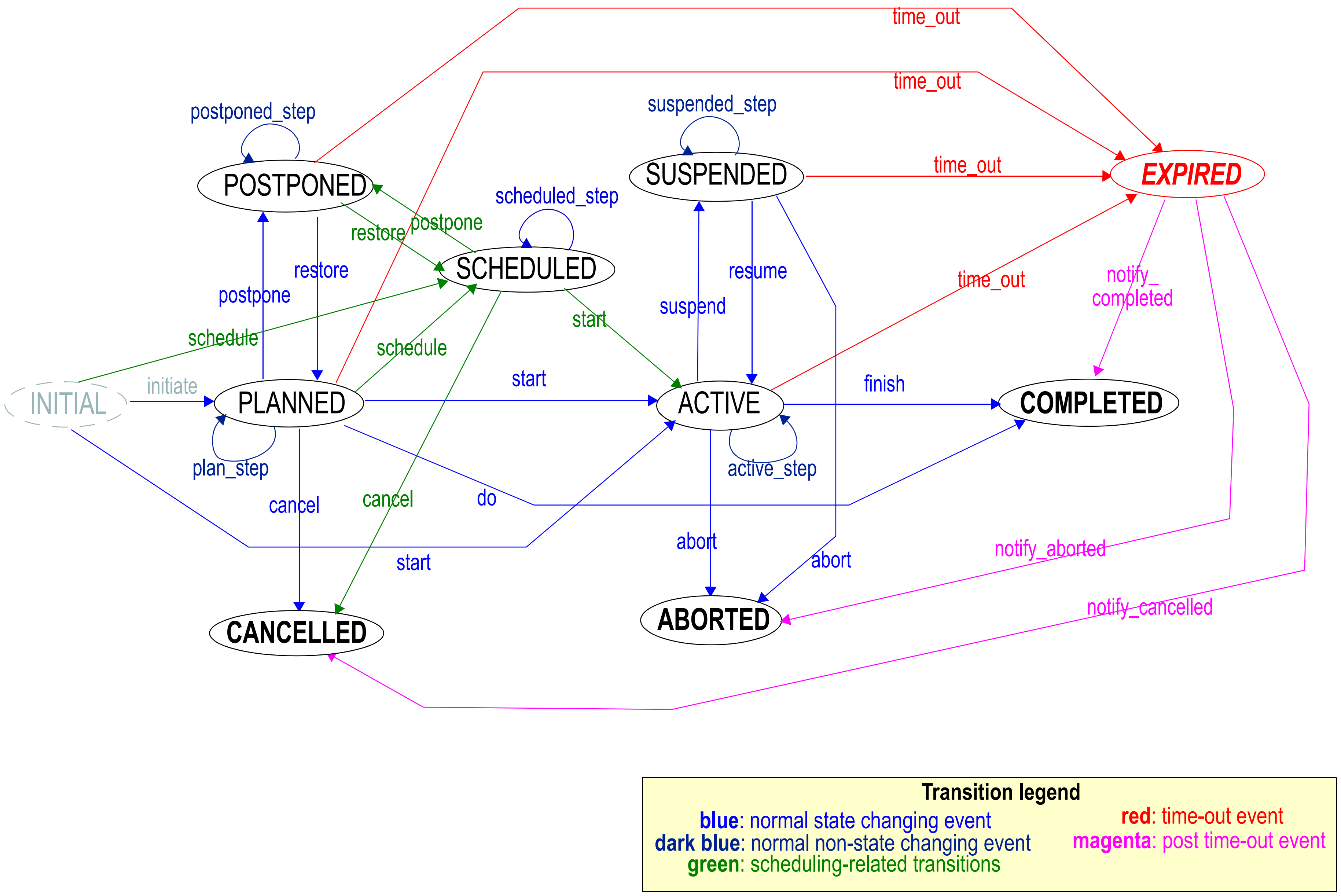


FIGURE 18 *open*EHR standard Instruction State Machine

- a way of knowing the state of any intervention, in terms of the states in a standard state machine, shown in FIGURE 18; this allows a patient's EHR to be queried in a standard way so as to return "all active medications", "all suspended interventions" etc;
- a way of mapping particular care process flow steps to the standard state machine states, enabling health professionals to define and view interventions in terms they understand;
- support for automated workflow, without requiring it.

Coupled with the comprehensive versioning capabilities of *open*EHR, the Instruction/Action design allows clinical users of the record to define and manage interventions for the patient in a distributed environment.

6.6 Time in the EHR

Time is well-known as a challenging modelling problem in health information. In *open*EHR, times that are a by-product of the investigation process (e.g. time of sampling or collection; time of measurement, time of a healthcare business event, time of data committal) described above are concretely modelled, while other times specific to particular content (e.g. date of onset, date of resolution) are modelled using archetyping of generic data attributes. The following figure shows a typical relationship of times with respect to the observation process, and the corresponding attributes within the *open*EHR reference model. Note that under different scenarios, such as GP consultation, radiology reporting and others, the temporal relationships may be quite different than those shown in the figure. Time is described in detail in the EHR Information Model document.

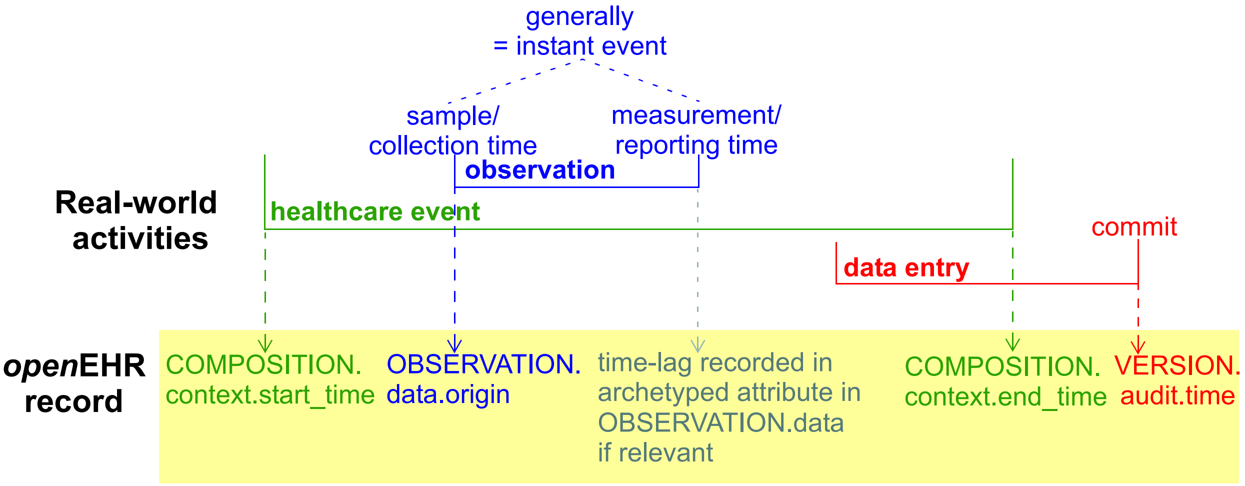


FIGURE 19 Time in the EHR

7 Security and Confidentiality

7.1 Requirements

Privacy, Confidentiality and Consent

Privacy (the right to limit who sees the personal data) and confidentiality (the obligation of others to respect the privacy of disclosed data) are primary concerns of many consumers with respect to e-Health systems. A widely accepted principle is that information provided (either directly or due to observation or testing of specimens etc) in confidence by a patient to health professionals during an episode of care should only be passed on or otherwise become available to other parties if the patient agrees; put more simply: *data sharing must be controlled by patient consent*. A more complex sub-requirement for some patients is allowing differential access to *parts* of their health record, for example, relatively open access rights to most of the health record, but limited access to sexual or mental health items. The interrelatedness of health information can make this difficult. For example the medication list will often give away sensitive conditions even if the diagnosis is hidden, yet is needed for any safe treatment, and many health professionals would see the unavailability of current medications (and allergies) information as highly problematic for giving even basic care.

Requirements of Healthcare Providers

On the other hand, clinical professionals delivering care want fast access to relevant data, and to be sure that what they see on the screen is a faithful representation of what has been said about the patient. Emergency access to health records is sometimes needed by carers otherwise unrelated to the normal care of a patient; such accesses can only be consented to in a general way, since the specific providers involved will not usually be known.

Researchers in healthcare generally want access to the data of large numbers of patients in order to evaluate current care and improve it, or for educational purposes. Both of these latter needs ultimately are patient and societal priorities also. Providing effective care and supporting ongoing medical research therefore have to function in a system that implements the concept of patient consent.

Specifying Access Control

In theory, it should be easy for the patient or some clinical professional to specify who can see the patient record. In some cases it is can be done by direct identification, e.g. the patient might nominate their long-term GP by provider id. Some exclusions could potentially be made this way as well, for example a previous doctor with whom the patient has a problematic personal relationship.

However it soon becomes difficult to identify provider parties individually when the patient moves into parts of the healthcare system where there are many staff, and/or where there is no previously established relationship. The advent of e-prescribing and e-pharmacy will bring even larger numbers of health and allied health workers into the e-Health matrix, making the problem of individual identification of who should see the patient's data unfeasible. Further, there is a large and growing category of "very mobile" people (the military, entertainers, NGO workers, international business and tourism professionals, athletes....) who cannot predict even in which country they may require care. As a consequence, the need for some access control to be specified in terms of categories or role types appears inescapable.

The Problem of Roles

One of the difficult challenges to implementing access control to health information is that of defining "roles", i.e. the status of users of the record at the time when right of access is being determined. In principle, roles ought to be knowable in advance. For example, the labels "nurse", "GP" and "psychiatrist" can be relatively easily assigned to individuals. However, the kinds of labels that are of more importance are those that differentiate among (for example) personal carers (e.g. primary GP), other care delivery staff (e.g. nurses, but also aged carers) and support staff (e.g. pathologists, radiographers). In a patient care delivery-oriented view of the world, the professional level of a health care professional is probably much less important than his or her relationship to the current care process for the patient.

It will not always be clear which individuals fall into any of these categories at any time, or how such terms are even defined in different sites and jurisdictions. Realistically, the evaluation of a role category such as "care deliverer" into particular identities such as those of nurses on the ward on a particular day must be done in the care delivery environment, not in the EHR. Access decisions for information in the EHR therefore will have some dependence on provider site knowledge of which staff are actively involved in the care process of a given patient.

Role-based access control is further complicated by the common fact of temporary replacements due to illness or holiday and role changes due to staff shortages. Further, if a physician employing a medical secretary requires her to access and update sensitive parts (relating to his own treatment of the patient) of the record, access at the highest level is effectively given to someone not medically trained or related directly to the patient's care, even if only for 10 minutes. Any role-based system therefore has to taken into account the messy reality of clinical care in the real world rather than being based solely on theoretical principles.

Usability

Usability of security and privacy mechanisms is a key requirement of a health record architecture. Some very elegant solutions to fine-grained access control designed by security experts would be simply unusable in practice because they would take too long for patients and doctors to learn, or are too time-consuming to actually use on the screen; they could also be too complex to safely implement in software.

The following sections describe support in *open*EHR for the main security and privacy requirements of EHRs.

7.2 Threats to Security

Any model of how security and privacy are supported in the health record must be based on some notion of assumed threats. Without going into great detail, security threats assumed by *open*EHR include (here "inappropriate" means anything that is not or would not be consented to by the patient):

- human error in patient identification, leading to incorrect association of health data of one patient with another. Mis-identification of patients can cause personal data to go into the record of another patient (leading to privacy violations and possibly clinical errors), or into a new record rather than the existing one for the same patient (leading to two or more clinically incomplete records);
- inappropriate access by health professionals or others in the physical care delivery environment (including e.g. any worker in a hospital) not related to the current care of the patient;
- inappropriate access by other persons known to the patient, e.g. father of a teenager accessing the child's health record;
- inappropriate access of health data by corporate or other organisations e.g. for purposes of insurance discrimination;
- malicious theft of or access to health data (e.g. of a celebrity or politician) for profit or other personal motives;

- generic threats to data integrity and availability, such as viruses, worms, denial of service attacks etc;
- failures in software (due to bugs, incorrect configuration, interoperability failures etc) causing corruption to data, or incorrect display or computation, resulting in clinical errors.

A key principle with respect to the design of mechanisms supporting security, confidentiality and integrity has to be kept in mind: the likelihood of any given mode of targetted inappropriate access is proportional to the perceived value of the information and inversely proportional to the cost of access. To paraphrase Ross Anderson's BMA paper[1] on health data security, for a given access, the perpetrator will try to find the simplest, cheapest and quickest method, which is more likely to be bribery or burglary than James Bond-inspired technology. *open*EHR makes use of this principle by providing some relatively simple mechanisms that are cheap to implement but can make misuse quite difficult, without compromising availability.

7.3 Solutions Provided by *open*EHR

7.3.1 Overview

Many of the concrete mechanisms relating to security and privacy are found in system deployments rather than in models such as *open*EHR, particularly the implementation of authentication, access control, and encryption. The *open*EHR specifications and core component implementations do not explicitly define many concrete mechanisms since there is great variability in the requirements of different sites - secure LAN deployments many require minimal security inside, whereas web-accessible health record servers are likely to have quite different requirements. What *open*EHR does is to support many of the key requirements in a flexible enough way that deployments with substantially different requirements and configurations can nevertheless implement the basic principles.

7.3.2 Security Policy

In and of itself, the *open*EHR EHR imposes only a minimal security policy profile which could be regarded as necessary, but generally not sufficient for a deployed system (i.e. other aspects would still need to be implemented in layers whose semantics are not defined *open*EHR). The following policy principles are embodied in *open*EHR.

Indelibility: health record information cannot be deleted; logical deletion is achieved by marking the data in such a way as to make it appear deleted (implemented in version control).

Anonymity: the content of the health record is separate from identifying demographic information. This can be configured such that theft of the EHR provides no direct clue to the identity of the owning patient (indirect clues are of course harder to control). Stealing an identified EHR involves theft of data from two servers, or even theft of two physical computers, depending on deployment configuration.

Access control: the overriding principle of access control must be "relevance" both in terms of user identity (who is delivering care to the patient) and time (during the current episode of care). An access control list can be created for the EHR, containing both identified individuals and categories (which might be role types, or particular staff groups). The

[1] see e.g. Ross Anderson - "Security in Clinical Information Systems" available at http://www.cl.cam.ac.uk/users/rja14/policy11/policy11.html.

patient must consent to all changes to the access control list; all changes to the list must be audit-trailed as for normal data.

To Be Determined: the detailed model of the access control list is not yet specified in openEHR, as there is currently no published model of access management for health data that has been shown to be usable and scalable. Various models are being investigated.

Audit trailing: All changes made to the EHR are audit-trailed with user identity, time-stamp, reason, optionally digital signature and relevant version information; one exception is where the modifier is the patient, in which case, a symbolic identifier can be used (known as `PARTY_SELF` in *open*EHR).

To Be Determined: *Patient privacy settings*: patients can mark Compositions in the EHR as having one of a number of levels of privacy. The definition of the privacy levels is not hard-wired in the *open*EHR models but rather is defined by standards or agreements within certain jurisdictions.

Other security policy items that should be implemented in even a minimal EHR deployment but are not directly specified by *open*EHR include the following.

Access logging: read accesses by application users to EHR data should be logged in the EHR system. Currently *open*EHR does not specify models of such logs, but might do so in the future. Studies have shown that making users aware of the fact of access logging is an effective deterrent to inappropriate access (especially where other controls are not implemented). There are some proponents of the argument that even read-access logs should be made part of the content of the EHR proper; currently *open*EHR does not support this approach.

Record demerging: when data for a patient is found to be in another patient's EHR, the access logs for that EHR should be used to determine who has accessed that data, primarily to determine if subsequent clinical thinking (e.g. diagnoses, medication decisions) have been made based on wrong information.

Record merging: when more than one EHR is discovered for the same patient, and have to be merged into a single record, the access control lists have to be re-evaluated and merged by the patient and potentially relevant carers.

Time-limitation of access: mechanisms should be implemented that limit the time during which given health professionals can see the patient record. Usually, the outer limits are defined by the interval of the episode of care in an institution plus some further time to cover follow-up or outpatient care. Episode start and end are recorded in *open*EHR by means of Admin_entries containing admission and discharge details.

Non-repudiation: if digital signing of changes to the record is made mandatory, non-repudiation of content can be supported by an *open*EHR system. The digital signing of communications (EHR Extracts) is also supported in *open*EHR; coupled with logging of communication of Extracts, this can be used to guarantee non-repudiation of information passed between systems (cf information passed between back-end and front-end applications of the same system).

A key feature of the policy is that it must scale to distributed environments in which health record information is maintained at multiple provider sites visited by the patient.

As Anderson points out in the BMA study, policy elements are also needed for guarding against users gaining access to massive numbers of EHRs, and inferencing attacks. Currently these are outside the scope of *open*EHR, and realistically, of most EHR implementations of any kind today.

The following sections describe how *open*EHR supports the first list of plicy objectives.

7.3.3 Integrity

Versioning

The most basic security-related feature of *open*EHR is its support for data integrity. This is mainly provided by the versioning model, specified in the `change_control` package in the Common Information Model, and in the Extract Information Model. Change-set based versioning of all information in the EHR and demographic services constitutes a basic integrity measure for information, since no content is ever physically modified, only new versions are created. All logical changes and deletions as well as additions are therefore physically implemented as new Versions rather than changes to existing information items. Clearly the integrity of the information will depend on the quality of the implementation; however, the simplest possible implementations (1 Version = 1 copy) can provide very good safety due to being write-once systems.

The use of change-sets, known as Contributions in *open*EHR, provides a further unit of integrity corresponding to all items modified, created or deleted in a single unit of work by a user.

The *open*EHR versioning model defines audit records for all changed items, which can be basic audits and/or any number of additional digitally signed attestations (e.g. by senior staff). This means that every write access of any kind to any part of an *open*EHR record is logged with the user identification, time, reason, and potentially other meta-data. Versioning is described in detail in section 8 on page 41.

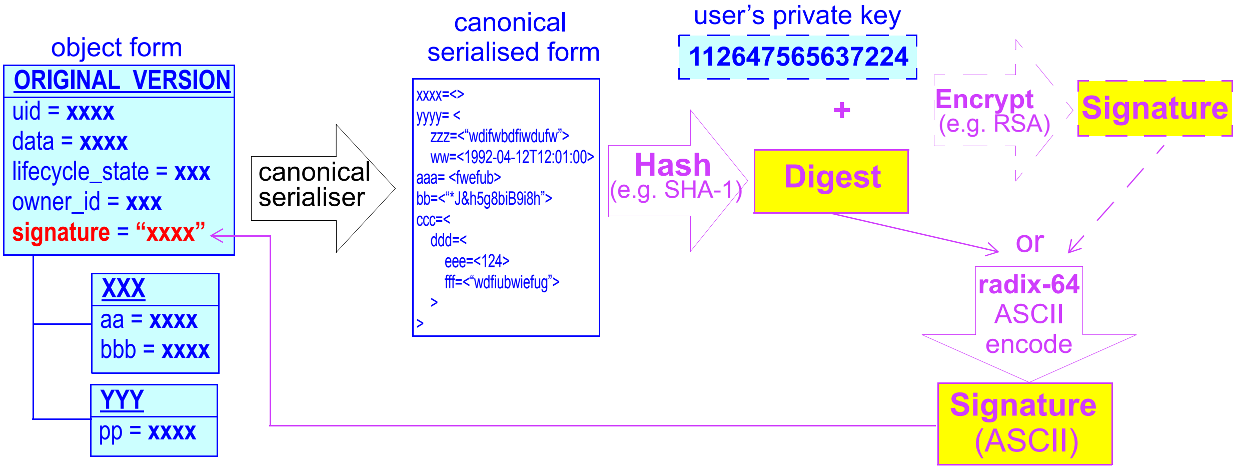
Digital Signature

The possibility exists within an *open*EHR EHR to digitally sign each Version in a Versioned object (i.e. for each Version of any logical item, such as medications list, encounter note etc). The signature is created as a private-key encryption (e.g. RSA-1) of a hash (e.g. MD5) of a canonical representation (such as in schema-based XML) of the Version being committed. A likely candidate for defining the signature and digest strings in *open*EHR is the openPGP message format (IETF RFC2440[1]), due to being an open specification and self-describing. The use of RFC2440 for the format does not imply the use of the PGP distributed certificate infrastucture, or indeed any certification infrastructure; *open*EHR is agnostic on this point. If no public key or equivalent infrastructure is available, the encryption step might be omitted, resulting in a digest only of the content. The signature is stored within the Version object, allowing it to be conveniently carried within EHR Extracts. The process is shown in the following figure.

The signing of data in a versioning system acts as an integrity check (the digest performs this function), an authentication measure (the signature performs this function), and also a non-repudiation measure. To guard against hacking of the versioned persistence layer itself, signatures can be forwarded to a trusted notarisation service. A fully secure system based on digital signing also requires certificated public keys, which may or may not be available in any given environment.

One of the benefits of digitally signing relatively small pieces of the EHR (single Versions) rather than the whole EHR or large sections of it is that the integrity of items is more immune to localised repository corruptions.

[1] IETF RFC 2440 - http://www.ietf.org/internet-drafts/draft-ietf-openpgp-rfc2440bis-18.txt



```
xxxx=<>
yyyy= <
   zzz=<"wdifwbdfiwdufw">
   ww=<1992-04-12T12:01:00>
aaa= <fwefub>
bb=<"*J&h5g8biB9i8h">
ccc=<
   ddd=<
      eee=<124>
      fff=<"wdfiubwiefug">
   >
>
```

FIGURE 20 Version Signature

7.3.4 Anonymity

As described above in section 6.1, one of the features of the *open*EHR EHR is a separation of EHR (clinical and administrative) information and demographic information. This mainly relates to references to the patient rather than to provider entities, since the latter are usually publicly known. A special kind of object known as `PARTY_SELF` in *open*EHR is used to refer to the subject in the EHR. The only information contained in a `PARTY_SELF` instance is an optional external reference. The *open*EHR EHR can be configured to provide 3 levels of separation by controlling whether and where this external identifier is actually set in `PARTY_SELF` instances, as follows:

- Nowhere in the EHR (i.e. every `PARTY_SELF` instance is a blank placeholder). This is the most secure approach, and means that the link between the EHR and the patient has to be done outside the EHR, by associating EHR.*ehr_id* and the subject identifier. This approach is more likely for more open environments.
- Once only in the EHR Status object (subject attribute), and nowhere else. This is also relatively secure, if the EHR Status object is protected in some way.
- In every instance of `PARTY_SELF`; this solution is reasonable in a secure environment, and convenient for copying parts of the record around locally.

This simple mechanism provides a basic protection against certain kinds of information theft or hacking if used properly. In the most secure situation, a hacker has to steal not just EHR data but also separate demographic records and an identity cross reference database, both of which can be located on different machines (making burglary harder). The identity cross-reference database would be easy to encrypt or protect by other security mechanisms.

7.4 Access Control

Overview

`To Be Continued:`

Privacy Setting

`To Be Continued:`

Access Control List

`To Be Continued:`

Emergency Override

There are clearly situations in which a patient requires care but which do not conform to the established definitions of consent (i.e. access control) in the EHR. These include not only accidents, but also incidents in aged care facilities and other places, where the available physicians are unknown to the patient; the patient may also be in a foreign environment and be unknown to the health system there. In such cases, if the patient is unconscious or other factors make the gaining of consent by the usual means impractical, emergency access, if technically possible (it may be that some or all of the data are encypted) will most likely override the established access control definitions. The main protection for the patient is the rigorous use of read access logging, described below.

Read Access Logging

One of the requirements of a secure health information environment is logging of read accesses. Studies have shown that this is an effective deterrent to minor (but probably frequent) misuses by staff of health data. It also provides support for forensic investigation of inappropriate access after the fact by people who have managed to obtain official clearance and therefore are not prevented by the system from seeing requested data. Access logging can be used in two main scenarios. The first is *in situ* access, usually by a user using a GUI application. The access log then records what data were delivered to an application and might have been seen by the user. The second scenario is when EHR data are sent to another system, e.g. as an EHR Extract. The sending of data to a nominated user should be logged locally. With the advent of web-services and large scale VPNs (virtual private networks), the distinction between these modes of sharing may well cease to be visible, meaning that logging for both should probably done in a similar if not the same way.

The *open*EHR specifications do not currently include definitions for read logging, but this may change in the future, depending on the need for interoperability of such data, e.g. with applications designed for auditing health data access.

Setting Access Control

How patients consent to the use of their own data is set is one of the greatest challenges of securing health data. Any access control system has to be easily *explainable* to both patients and health professionals (who are not *a priori* likely to be any better using computers than patients), and efficiently usable. Both the simplicity of the access control model (e.g. sensitivity levels and/or some access control lists) and the simplicity of the software via which consent is set are crucial to any scheme being workable. In particular, patients and physicians are unlikely to want to have to decide for each piece of information what the access list and sensitivity is. Consequently, for many patients a semi-automatic means of setting consent will be required with manual setting probably limited to very small amounts of very sensitive information.

Automating the setting of consent may be possible with a classification system for information items that is comprehensible by the patient, such as categories like “mental health”, “sexual health”, and “family planning”. In *open*EHR, this is potentially provided by archetypes, which ensure that the clinical type of information of each chunk of information is known. However, the correspondence between the type of information and the purpose of recording it is not always obvious; a further complication is that basic information like the medications and allergy lists need to be widely visible, but can easily give away sensitive health conditions.

Whether a workable solution to these issues will become available is not currently clear. Future additions to *open*EHR that support solutions in thie area will be in favour of usability and protection of the most sensitive information.

7.4.1 Pseudonymisation

At one level, pseudonymisation of *open*EHR data for research purposes is easy to achieve, due to the division of EHR and demographic data. It is easy to process an *open*EHR record and completely remove all direct mention of the patient. However, depending on the kind of study being done, two problems remain. Firstly, the clinical record is likely to contain some amount of identifying information (e.g. patient is a double-amputee, has a rare condition, or is recorded living near or in an identified location), making it a target of an “inference” attack. Secondly, some demographic data for the patient (address, occupation, ethnicity, religion etc) may be medically significant.

Converting an *open*EHR record to acceptable pseudonymised form for a given study is facilitated by the use of archetypes and archetype-based querying. Because it is easy to find out what archetypes have been used in a record, and what archetypes are relevant to a study, it is not hard to formalise and implement a way of filtering record content to generate a pseudonymised, *relevant* projection. Further processing may still be required if it is thought that inappropriate identifying information may have been recorded in narrative text fields.

8 Versioning

8.1 Overview

Version control is an integral part of the *open*EHR architecture. An *open*EHR repository for EHR or demographic information is managed as a change-controlled collection of “version containers” (modelled by the `VERSIONED_OBJECT<T>` class in the `common.change_control` package), each containing the versions of a top-level content structure (such as a Composition or Party) as it changes over time. A version-controlled top-level content structure is visualised in FIGURE 21.

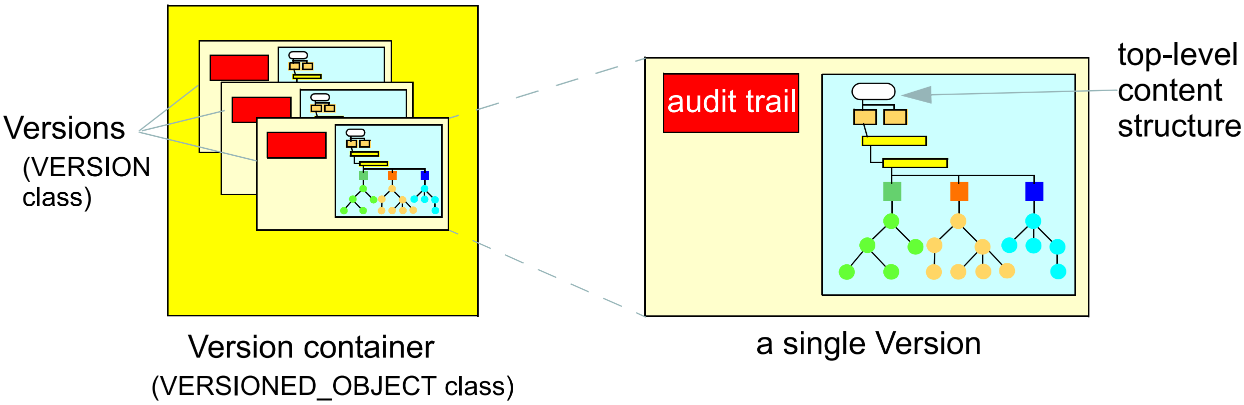


FIGURE 21 Version-control structures

Versioning of single top-level structures is a necessary, but not sufficient requirement for a repository that must provide coherence, traceability, indelibility, rollback, and support for forensic examination of past states of the data. Features supporting “change control” are also required. Under a disciplined change control scheme, changes are not made arbitrarily to single top-level structures, but to the repository itself. Changes take the form of change-sets, called “contributions”, that consist of new or changed versions of the controlled items in the repository. The key feature of a change-set is that it acts like a transaction, and takes the repository from one consistent state to another, whereas arbitrary combinations of changes to single controlled items coould easily be inconsistent, and even dangerously wrong where clinical data are concerned.

These concepts are well-known in configuration management (CM), and are used as the basis for most software and other change management systems, including numerous free and commercial products available today. They are a central design feature of *open*EHR architecture. The following sections provide more details

8.2 The Configuration Management Paradigm

The “configuration management” (CM) paradigm is well-known in software engineering, and has its own IEEE standard[1]. CM is about managed control of changes to a repository of items (formally called “configuration items” or CIs), and is relevant to any logical repository of distinct information items which changes in time. In health information systems, at least two types of information require such management: electronic health records, and demographic information. In most analyses in the past, the need for change management has been expressed in terms of specific requirements for audit trailing of changes, availability of previous states of the repository and so on. In *open*EHR, the aim is

[1] IEEE 828-2005 - standard for Software Configuration Management Plans.

to provide a formal, general-purpose model for change control, and show how it applies to health information.

8.2.1 Organisation of the Repository

The general organisation of a repository of complex information items such as a software repository, or the EHR consists of the following:

- a number of distinct information items, or *configuration items*, each of which is uniquely identified, and may have any amount of internal complexity;
- optionally, a directory system of some kind, in which the configurations items are organised;
- other environmental information which may be relevant to correctly interpreting the primary versioned items, e.g. versions of tools used to create them.

In a software or document repository, the CIs are files arranged in the directories of the file system; in an EHR based on *open*EHR, they are Compositions, the optoinal Folder structure, Parties in the demographic service and so on. Contributions are made to the repository by users. This general abstraction is visualised in FIGURE 22.

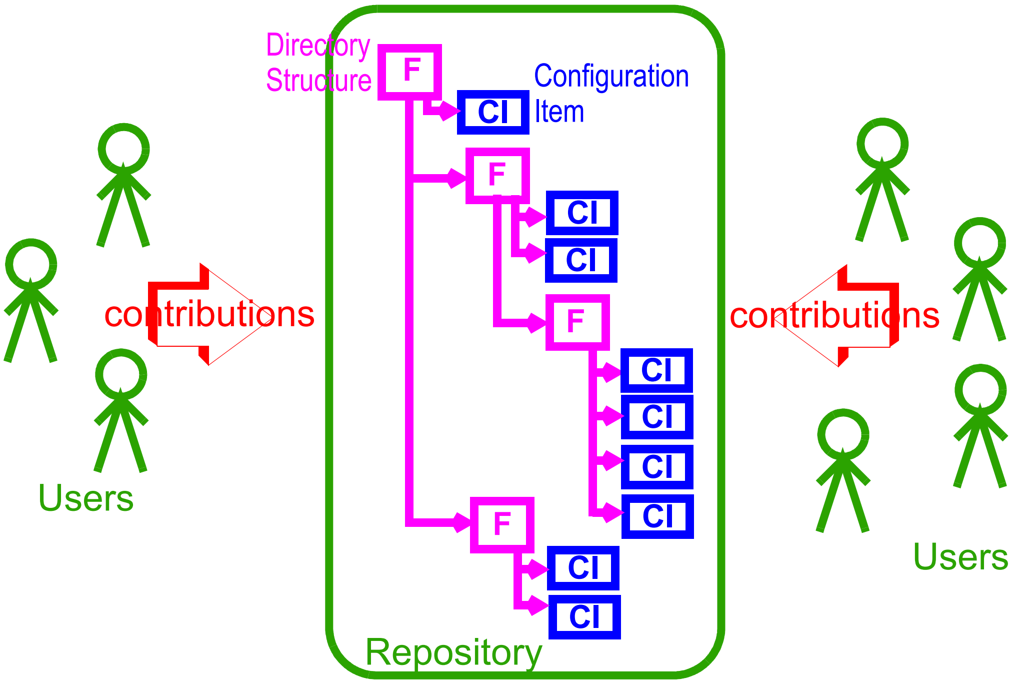


FIGURE 22 General Structure of a Controlled Repository

8.2.2 Change Management

Change doesn't occur to CIs in isolation, but to the repository as a whole. Possible types of change include:

- creation of a new CI;
- removal of a CI;
- modification of a CI;
- creation of, change to or deletion of part of the directory structure;
- moving of a CI to another location in the directory structure.

The goal of configuration management is to ensure the following:

- the repository is always in a valid state;
- any previous state of the repository can be reconstructed;
- all changes are audit-trailed.

8.3 Managing Change in Time

Properly managing changes to the repository requires two mechanisms. The first, *version control*, is used to manage versions of each CI, and of the directory structure if there is one. The second is the concept of the “change-set”, known as a *contribution* in *open*EHR. This is the *set* of changes to individual CIs (and the directory structure) made by a user as part of some logical change. For example, in a document repository, the logical change might be an update to a document that consists of multiple files (CIs). There is one contribution, consisting of changes to the document file CIs, to the repository. In the EHR, a contribution might consist of changes to more than one Composition, and possibly to the organising Folder structure.

A typical sequence of changes to a repository is illustrated below. FIGURE 23 shows a the effect of four Contributions (indicated by blue ovals on the left hand side) to a repository containing a number of CIs (that the directory tree is not shown for the sake of simplicity). As each Contribution is made, the repository is changed in some way. The first brings into existing a new CI, and modifies three others (changes indicated by the ‘C’ triangles). The second contribution causes the creation of a new CI only. The third causes a creation as well as two changes, while the fourth causes only a change. (Changes to the folder structure are not shown here).

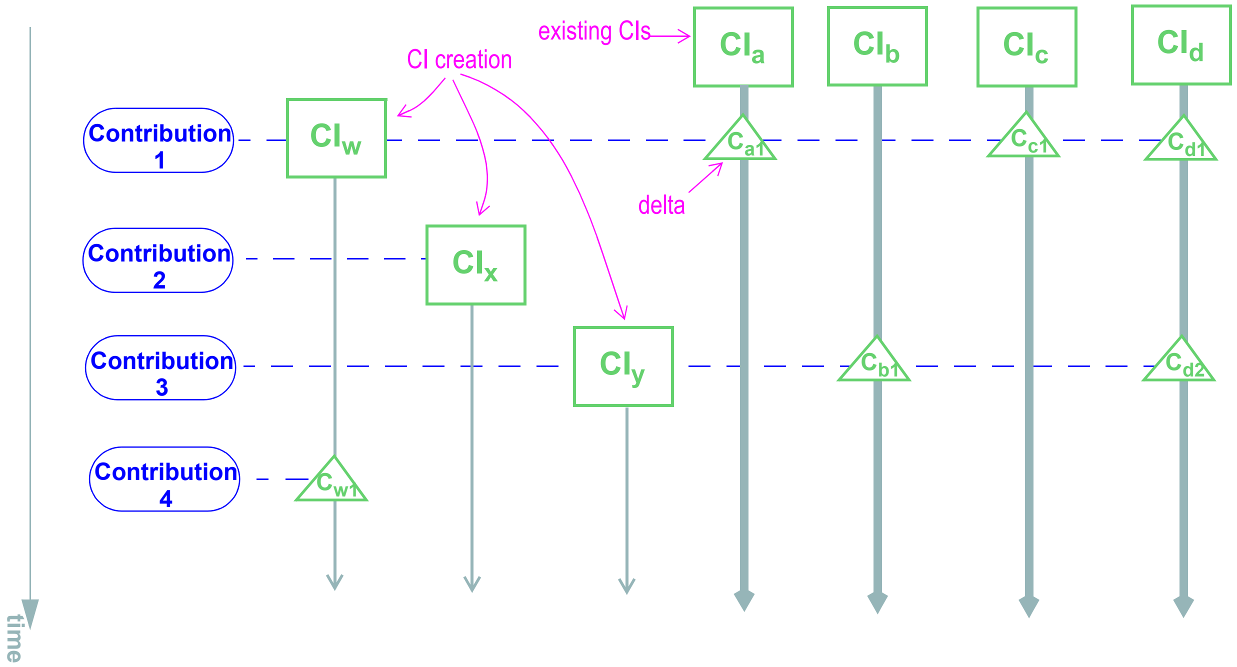


FIGURE 23 Contributions to the Repository (delta form)

One nuance which should be pointed out is that in FIGURE 23, contributions are shown as if they are literally a set of deltas, i.e. exactly the changes which occur to the record. Thus, the first contribution is the set $\{CI_w, C_{a1}, C_{c1}, C_{d1}\}$ and so on. Whether this is literally true depends on the construction of applications. In some situations, some CIs may be updated by the user viewing the current list and entering just the changes - the situation shown in FIGURE 23; in others, the system may provide the current state of these CIs for editing by the user, and submit the updated versions, as shown in FIG-

URE 24. Some applications may do both, depending on which CI is being updated. The internal versioning implementation may or may not generate deltas as a way of efficient storage.

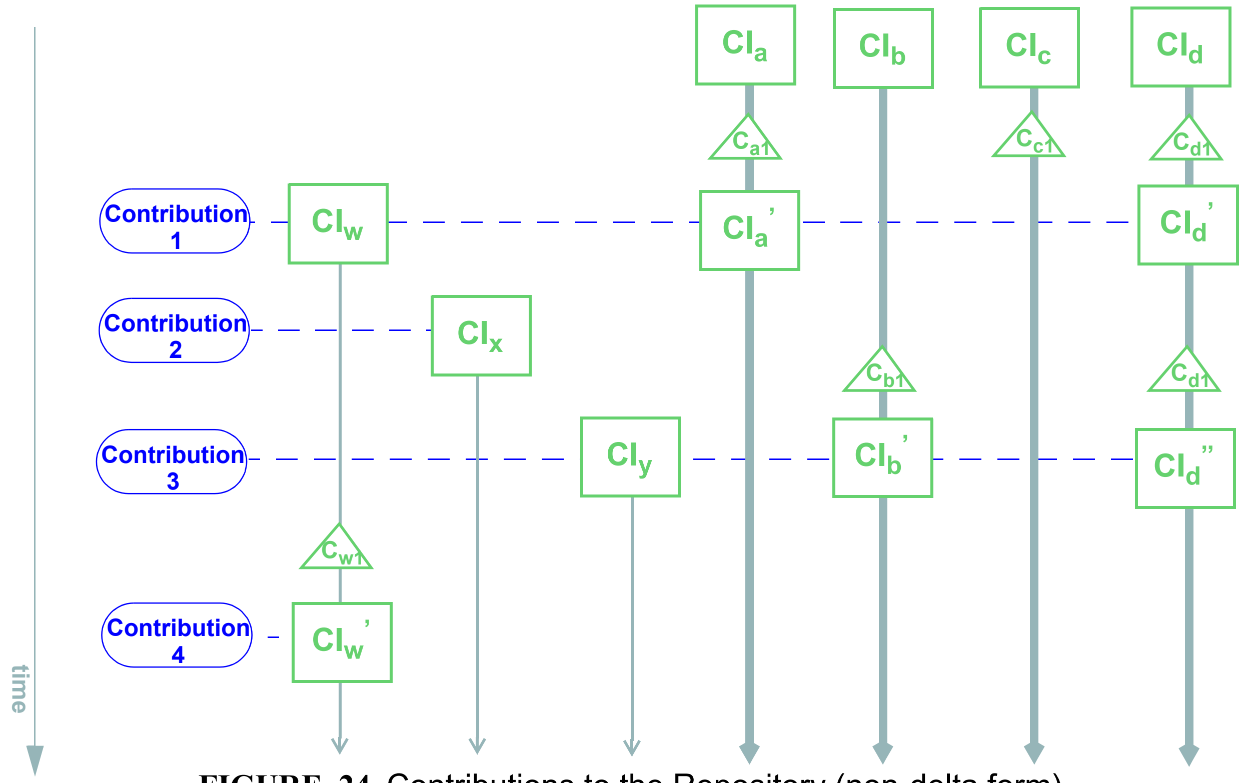


FIGURE 24 Contributions to the Repository (non-delta form)

For the purposes of *open*EHR, a contribution is considered as being the logical set of CIs changed or created at one time, as implied by FIGURE 24.

8.3.1 General Model of a Change-controlled Repository

FIGURE 25 shows an abstract model of a change-controlled repository, which consists of:

- version-controlled configuration items - instances of VERSIONED_OBJECT<T>;
- CONTRIBUTIONs;
- an optional directory system of of folders. If folders are used, the folder structure must also be versioned as a unit.

The actual type of links between the controlled repository and the other entities might vary - in some cases it might be association, in others aggregation; cardinalities might also vary. FIGURE 25 therefore provides a guide to the definition of actual controlled repositories, such as an EHR, rather than a formal specification for them.

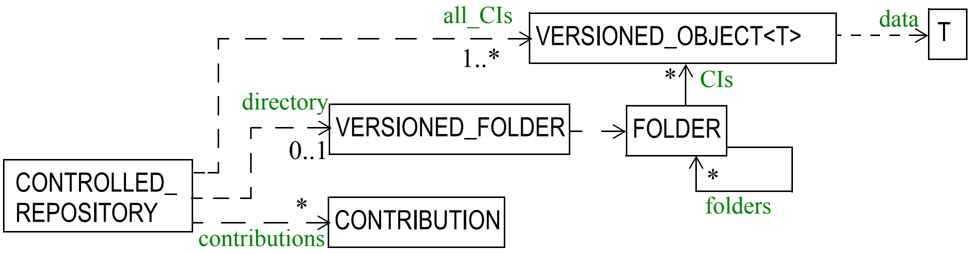


FIGURE 25 Abstract Model of Change-controlled Repository

8.4 The “Virtual Version Tree”

An underlying design concept of the versioning model defined in *open*EHR is known as a “virtual version tree”. The idea is simple in the abstract. Information is committed to a repository (such as an EHR) in lumps, each lump being the “data” of one Version. Each Version has its place within a version tree, which in turn is maintained inside a Versioned object (or “version container”). The virtual version tree concept means that any given Versioned object may have numerous copies in various systems, and that the creation of versions in each is done in such a way that all versions so created are in fact compatible with the “virtual” version tree resulting from the superimposition of the version trees of all copies. This is achieved using simple rules for version identification and is done to facilitate data sharing. Two very common scenarios are served by the virtual version tree concept:

- longitudinal data that stands as a proxy for the state or situation of the patient such as “Medications” or “Problem list” (persistent Compositions in *open*EHR) is created and maintained in one or more care delivery organisations, and shared across a larger number of organisations;
- some EHRs in an EHR server in one location are mirrored into one or more other EHR servers (e.g. at care providers where the relevant patients are also treated); the mirroring process requires asynchronous synchronisation between servers to work seamlessly, regardless of the location, time, or author of any data created.

The versioning scheme used in *open*EHR guarantees that no matter where data are created or copied, there are no inconsistencies due to sharing, and that logical copies are explicitly represented. It therefore provides direct support for shared data in a shared care context.

9 Identification

9.1 General Scheme

The identification scheme described here requires two kinds of "identifier": identifiers proper and references, or locators. An *identifier* is a unique (within some context) symbol or number given to an object, and usually written into the object, whereas a *reference* is the use of an identifier by an exterior object, to refer to the object containing the identifier in question. This distinction is the same as that between primary and foreign keys in a relational database system.

In the *open*EHR RM, identifiers and references are implemented with two groups of classes defined in the `support.identification` package. Identifiers of various kinds are defined by descendant classes of `OBJECT_ID`, while references are defined by the classes inheriting from `OBJECT_REF`. The distinction is illustrated in FIGURE 26. Here we see two container objects with `OBJECT_ID`s (since `OBJECT_ID` is an abstract type, the actual type will be another `XXX_ID` class), and various `OBJECT_REF`s as references.

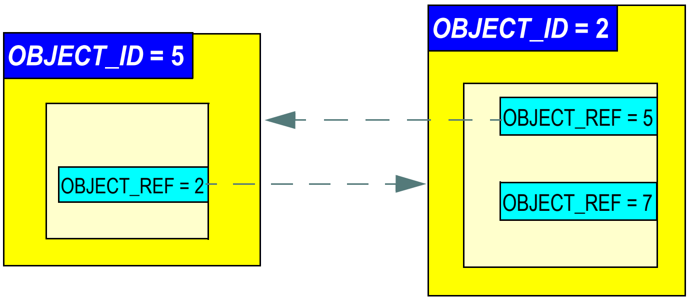


FIGURE 26 XXX_IDs and XXX_REFs

9.2 Levels of Identification

In order to make data items locatable from the outside, identification is supported at 3 levels in *open*EHR, as follows:

- *repository objects*: entities such as the EHR (EHR IM) and `VERSIONED_OBJECT`s (Common IM) are identified uniquely;
- *top-level content structures*: versioned content structures such as `COMPOSITION`, `EHR_STATUS`, `PARTY` etc are uniquely identified by the association of the identifier of their containing `VERSIONED_OBJECT` and the identifier of their containing `VERSION` within the container;
- *internal nodes*: nodes within top-level structures are identified using paths.

Three kinds of identification are used respectively. For repository structures, meaningless unique identifiers ("uids") are used. In most cases, the type `HIER_OBJECT_ID` will be used, which contains an instance of a subtype of the `UID` class, i.e. either an ISO OID or a IETF UUID (see http://www.ietf.org/rfc/rfc4122.txt; also known as a GUID). In general UUIDs are favoured since they require no central assignment and can be generated on the spot. A repository object can be referenced with an `OBJECT_REF` containing its identifier.

Versions of top-level structures are identified in a way that is guaranteed to work even in distributed environments where copying, merging and subsequent modification occur. The full identification of a version of a top-level structure is the globally unique tuple consisting of the *uid* of the owning

VERSIONED_OBJECT, and the two VERSION attributes *version_tree_id* and *creating_system_id*. The *version_tree_id* is a 1 or 3-part number string, such as “1” or for a branch, “1.2.1”. The *creating_system_id* attribute carries a unique identifier for the system where the content was first created; this may be a GUID, Oid or reverse internet identifier. A typical version identification tuple is as follows:

```
F7C5C7B7-75DB-4b39-9A1E-C0BA9BFDBDEC        -- id of VERSIONED_COMPOSITION
au.gov.health.rdh.ehr1                      -- id of creating system
2                                           -- current version
```

This 3-part tuple is known as a “Version locator” and is defined by the class OBJECT_VERSION_ID in the support.identification package. A VERSION can be *referred to* using a normal OBJECT_REF that contains a copy of the version’s OBJECT_VERSION_ID. The *open*EHR version identification scheme is described in detail in the change_control package section of the Common IM.

The last component of identification is the path, used to refer to an interior node of a top-level structure identified by its Version locator. Paths in *open*EHR follow an Xpath style syntax, with slight abbreviations to shorten paths in the most common cases. Paths are described in detail below.

To refer to an interior data node from outside a top-level structure, a combination of a Version locator and a path is required. This is formalised in the LOCATABLE_REF class in the change_control package section of the Common IM. A Universal Resource Identifier (URI) form can also be used, defined by the data type DV_EHR_URI (Data types IM). This type provides a single string expression in the scheme-space “ehr://” which can be used to refer to an interior data node from anywhere (it can also be used to represent queries; see below). Any LOCATABLE_REF can be converted to a DV_EHR_URI, although not all DV_EHR_URIs are LOCATABLE_REFs.

FIGURE 27 summarises how various types of OBJECT_ID and OBJECT_REF are used to identify objects, and to reference them from the outside, respectively.

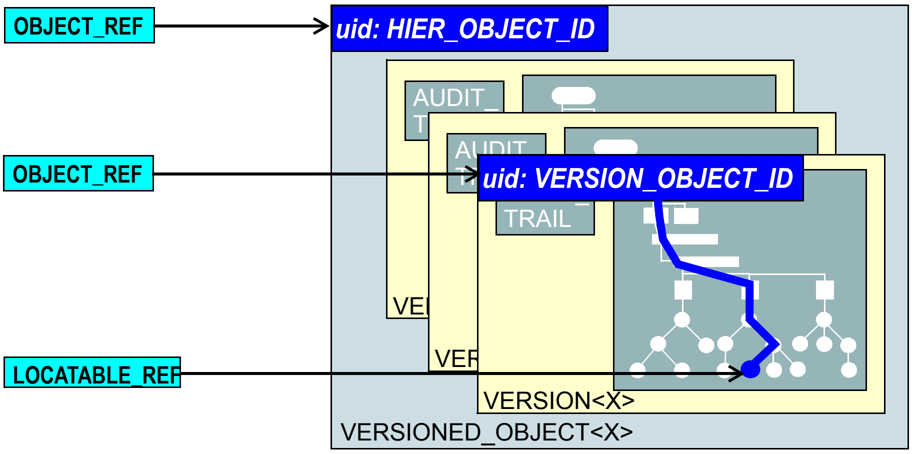


FIGURE 27 How to reference various levels of object

10 Archetyping

10.1 Overview

Under the two-level modelling approach, the formal definition of information structuring occurs at two levels. The lower level is that of the reference model, a stable object model from which software and data can be built. Concepts in the *open*EHR reference model are invariant, and include things like Composition, Section, Observation, and various data types such as Quantity and Coded text. The upper level consists of domain-level definitions in the form of *archetypes* and *templates*. Concepts defined at this level include things such as “blood pressure measurement”, “SOAP headings”, and “HbA1c Result”.

Archetypes are themselves instances of an *archetype model*, which defines a language in which to write archetypes. Archetypes are general-purpose, re-usable, and composable. They are used at runtime by building *templates* from them. A template is a tree of archetypes each of which constrains instances of various types in the reference model, i.e. Compositions, Section hierarchies, Entries and so on. Thus, while there are likely to be archetypes for such things as “biochemistry results” (an Entry archetype) and “SOAP headings” (a Section archetype), templates are used to put archetypes together to form whole Compositions in the EHR, e.g. for “discharge summary”, “antenatal exam” and so on. Templates correspond closely to screen forms and printed reports.

A template is used at runtime to create default data structures and to validate data input, ensuring that all data in the EHR conform to the constraints defined in the archetypes comprising the template. In particular, it conforms to the *path* structure of the archetypes, as well as their terminological constraints. Which archetypes were used at data creation time is written into the data, in the form of both archetype identifiers at the relevant root nodes, and archetype node identifiers - normative node names, which are the basis for paths. When it comes time to modify or query data, these archetype data enable applications to retrieve and use the original archetypes, ensuring modifications respect the original constraints, and allowing queries to be intelligently constructed.

All information conforming to the *open*EHR Reference Model (RM) - i.e. the collection of Information Models (IMs) - is “archetypable”, meaning that the creation and modification of the content, and subsequent querying of data is controllable by archetypes. Archetypes are themselves separate from the data, and are stored in their own repository. The archetype repository at any particular location will usually include archetypes from well-known online archetype libraries. Archetypes are deployed at runtime via templates that specify particular groups of archetypes to use for a particular purpose, often corresponding to a screen form.

10.2 Scope of Archetypes and Templates

All nodes within the top-level information structures in the *open*EHR RM are “archetypable”, with certain nodes within those structures being archetype “root points”. Each top-level type is always guaranteed to be an archetype root point. Although it is theoretically possible to use a single archetype for an entire top-level structure, in most cases, particularly for `COMPOSITION` and `PARTY`, a hierarchical structure of multiple archetypes will be used. This allows for componentisation and reusability of archetypes. When hierarchies of archetypes are used for a top-level structure, there will also be archetype root points in the interior of the structure. For example, within a `COMPOSITION`, `ENTRY` instances (i.e. `OBSERVATIONs`, `EVALUATIONs` etc) are almost always root points. `SECTION` instances are root points if they are the top instance in a Section structure; similarly for `FOLDER` instances within a directory structure. Other nodes (e.g. interior `SECTIONs`, `ITEM_STRUCTURE` instances) might also be archetype root points, depending on how archetypes are applied at runtime to data. FIGURE 28 illustrates the application of archetypes and templates to data.

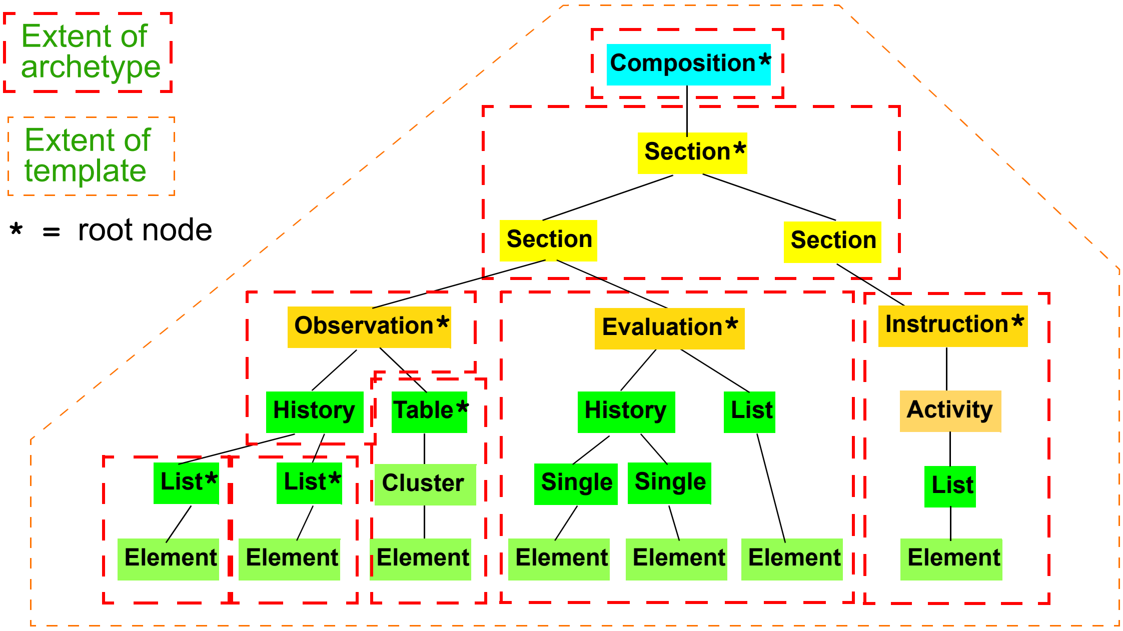


FIGURE 28 How Archetypes apply to Data

10.3 Archetype-enabling of Data

Archetype-enabling is achieved via inheritance into all concrete types in the RM of the class LOCATABLE from the package common.archetyped (see Common IM). The LOCATABLE class includes the attributes *archetype_node_id* and *archetype_details*. In the data, the former carries an identifier from the archetype. If the node in the data is a root point, it carries the multipart identifier of the generating archetype, and *archetype_details* carries an ARCHETYPED object, containing information pertinent to archetype root points. If it is a non-root node, the *archetype_node_id* attibute carries the identifier (known as an "at", or "archetype term" code) of the archetype interior node that generated the data node, and the *archetype_details* attribute is void.

Sibling nodes in data can carry the same *archetype_node_id* in some cases, since archetypes provide a pattern for data, rather than an exact template. In other words, depending on the archetype design, a single archetype node may be replicated in the data.

In this way, each *archetyped data composition*[1] in *open*EHR data has a generating archetype which defines the particular configuration of instances to create the desired composition. An archetype for "biochemistry results" is an OBSERVATION archetype, and constrains the particular arrangement of instances beneath an OBSERVATION object; a "problem/SOAP headings" archetype constrains SECTION objects forming a SOAP headings structure. In general, an archetyped data composition is any composition of data starting at a root node and continuing to its leaf nodes, at which point lower-level compositions, if they exist, begin. Each of the archetyped areas and its subordinate archetyped areas in FIGURE 28 is an archetyped data composition.

[1] Note: care must be taken not to confuse the general term "composition" with the specific use of this word in *open*EHR and CEN EN 13606, defined by the COMPOSITION class; the specific use is always indicated by using the term "Composition".

The result of the use of archetypes to create data in the EHR (and other systems) is that the structure of data in any top-level object conforms to the constraints defined in a composition of archetypes chosen by a template, including all optionality, value, and terminology constraints.

10.4 Archetypes, Templates and Paths

The use of archetypes and templates enables paths to be used ubiquitously in the *open*EHR architecture. Archetypes and templates have their own paths, constructed from attribute names and archetype node identifiers, in an Xpath-compatible syntax. Thes paths serve to identify any node in a template or archetype, such as the "diastolic blood pressure" `ELEMENT` node, deep within a "blood pressure measurement" archetype. Since archetype node identifiers are embedded into data at runtime, archetype paths can be used to extract data nodes conforming to particular parts of archetypes, providing a very powerful basis for querying. "Runtime" paths can also be constructed in data, consisting of more complex predicates (still in the Xpath style). Paths in *open*EHR are explained in detail under Paths and Locators on page 51.

11 Paths and Locators

11.1 Overview

The *open*EHR architecture includes a path mechanism that enables any node within a top level structure to be specified from the top of the structure. The combination of a path and a Version identifier such as `OBJECT_VERSION_ID` forms a "globally qualified node reference" which can be expressed using `LOCATABLE_REF`. It can also be expressed in portable URI form as a `DV_EHR_URI`, known as a "globally qualified node locator". Either representation enables any *open*EHR data node to be referred to from anywhere. This section describes the syntax and semantics of paths, and of the URI form of reference.

11.2 Paths

11.2.1 Basic Syntax

Paths in *open*EHR are defined in an Xpath[1]-comptabile syntax which is a superset of the path syntax described in the Archetype Definition Language (ADL). The syntax is designed to be easily mappable to Xpath expressions, for use with *open*EHR-based XML.

The runtime path syntax used in locator expressions follows the general pattern of a path consisting of segments each consisting of an attribute name[2], and separated by the slash ('/') character, i.e.:

```
attribute_name / attribute_name / ... / attribute_name
```

Paths select the object which is the value of the final attribute name in the path, when going from some starting point in the tree and following attribute names given in the path. The starting point is indicated by the initial part of the path, and can be specified in three ways:

relative path: path starts with an attribute name, and the starting point is the current point in the tree (given by some previous operation or knowledge);

absolute path: path starts with a '/'; the starting point is the top of the structure;

movable path: path starts with a movable path leader '//' and is taken to be a pattern which can start anywhere in the data; the pattern is matched if an actual path can be found anywhere in the structure that matches the path given after the '//' leader.

11.2.2 Predicate Expressions

Paths specified solely with attribute names are limited in two ways. Firstly, they can only locate objects in structures in where there are no containers, such as lists or sets. However, in any realistic data, including most *open*EHR data, list, set and hash structures are common. Additional syntax is needed to match a particular object from among the siblings referred to by a container attribute. This takes the form of a predicate expression enclosed in brackets ('[]') after the relevant attribute in a segment, i.e.:

```
attribute_name [predicate expression]
```

The general form of a path then resembles the following:

[1] See W3C Xpath 1.0 specification, 1999. Available at http://www.w3.org/TR/xpath.

[2] In all openEHR documentation, the term "attribute" is used in the object-oriented sense of "property of an object", not in the XML sense of named values appearing within a tag. The syntax described here should not be considered to necessarily have a literal mapping to XML instance, but rather to have a logical mapping to object-oriented data structures.

```
attribute_name / attribute_name [predicate expression] / ...
```

Here, predicate expressions are used optionally on those attributes defined in the reference model to be of a container type (i.e. having a cardinality of > 1). If a predicate expression is not used on a container attribute, the whole container is selected.

The second limitation of basic paths is that they cannot locate objects based on other conditions, such as the object having a child node with a particular value. To address this, predicate expressions can also be used to select an object on the basis of other conditions relative to the object, by including boolean expressions including paths, operators, values and parentheses.

The syntax of predicate expressions used in *open*EHR is a subset of the Xpath syntax for predicates with a small number of shortcuts. The general form of a predicate statement is a boolean-returning expression consisting of paths, values, operators and parentheses. In the current release of *open*EHR, it is expected that only very simple expressions will be used. The simplest such expression is to identify an object by its *archetype_node_id* value, which will be an 'at' code from an archetype; in other words, just to use the ADL archetype path against the runtime data. A typical ADL path is the following (applied to an Observation instance):

```
/data/events[at0003]/data/items[at0025]/value/magnitude
```

This path refers to the magnitude of a 1-minute Apgar total in an Observation containing a full Apgar result structure. In this path, the `[atNNNN]` predicates correspond to `[@archetype_node_id = "atNNNN"]` in standard Xpath, however, the shorthand form is used in *open*EHR as it is the only kind of predicate used in archetype paths. In *open*EHR runtime paths, archetype code predicates are also commonly used, and the same shortcut is allowed. However, runtime path predicates can also include other expressions (including the orthodox Xpath equivalent expression for the archetype node id shortcut), typically based on the value of some other attribute such as `ELEMENT`.*name* or `EVENT`.*time*. Combinations of the *archetype_node_id* and other such values are likely to be commonly used in querying, such as the following path fragment (applied to an `OBSERVATION` instance):

```
/data/events[at0007 AND time >= "24-06-2005 09:30:00"]
```

This path would choose Events in Observation.data whose *archetype_node_id* meaning is "summary event" (at0007 in some archetype) and which occurred at or after the given time. The following example would choose an Evaluation containing a diagnosis (at0002.1) of "other bacterial intestinal infections" (ICD10 code A04):

```
/data/items[at0002.1
        AND value/defining_code/terminology_id/value = "ICD10AM"
        AND value/defining_code/code_string = "A04"]
```

11.2.3 Paths within Top-level Structures

Paths within top-level structures strictly adhere to attribute and function names in the relevant parts of the reference model. Predicate expressions are needed to distinguish multiple siblings in various points in paths into these structures, but particularly at archetype "chaining" points. A chaining point is where one archetype takes over from another as illustrated in FIGURE 28. Chaining points in Compositions occur between the Composition and a Section structure, potentially between a Section structure and other sub-Section structures (constrained by a different Section archetype), and between either Compositions or Section structures, and Entries. Chaining might also occur inside an Entry, if archetyping is used on lower level structures such as Item_lists etc. Most chaining points correspond to container types such as `List<T>` etc, e.g. `COMPOSITION`.*content* is defined to be a `List<CONTENT_ITEM>`, meaning that in real data, the content of a Composition could be a List of Section structures. To distinguish between such sibling structures, predicate expressions are used, based on the *archetype_id*. At the root point of an archetype in data (e.g. top of a Section structure), the *archetype_id* carries the identifier of the archetype used to create that structure, in the same manner as any interior point in an archetyped structure has an *archetype_node_id* attribute carrying archetype *node_id* values. The chaining point between Sections and Entries works in the same manner, and since multiple Entries can occur under a single Section, *archetype_id* predicates are also used to distinguish them. The same shorthand is used for *archetype_id* predicate expressions as for *archetype_node_ids*, i.e. instead of using `[@archetype_id = "xxxxx"]`, `[xxxx]` can be used instead.

The following paths are examples of referring to items within a Composition:

```
/content[openEHR-EHR-SECTION.vital_signs.v1]/
       items[openEHR-EHR-OBSERVATION.heart_rate-pulse.v1]/data/
       events[at0003 AND time='2006-01-25T08:42:20']/data/items[at0004]
/content[openEHR-EHR-SECTION.vital_signs.v1]/
       items[openEHR-EHR-OBSERVATION.blood_pressure.v1]/data/
       events[at0006 AND time='2006-01-25T08:42:20']/data/items[at0004]
/content[openEHR-EHR-SECTION.vital_signs.v1]/
       items[openEHR-EHR-OBSERVATION.blood_pressure.v1]/data/
       events[at0006 AND time='2006-01-25T08:42:20']/data/items[at0005]
```

Paths within the other top level types follow the same general approach, i.e. are created by following the required attributes down the hierarchy.

11.2.4 Runtime Paths and Uniqueness

Archetype paths are not guaranteed to be unique in data. However it is often necessary to be able to construct a unique path to any data item in real data. This can only be reliably done by using attributes other than *archetype_node_id*. Consider as an example the following `OBSERVATION` archetype:

```
OBSERVATION[at0000] matches {   -- blood pressure measurement
       data matches {
          HISTORY matches {
             events {1..*} matches {
                EVENT[at0001] {0..1}  matches {  -- any event
                   name matches {DV_TEXT matches {...}}
                   data matches {
                      ITEM_LIST matches {        -- systemic arterial BP
                         count matches {2..*}
                         items matches {
                            ELEMENT[at1100] matches {-- systolic BP
                               name matches {DV_TEXT matches {...}}
                               value matches {magnitude matches {...}}
                            }
                            ELEMENT[at1200] matches {-- diastolic BP
                               name matches {DV_TEXT matches {...}}
                               value matches {magnitude matches {...}}
                            }
                         }
                      }
                   }
                }
             }
          }
```

The following archetype path refers to the systolic blood pressure magnitude:

```
/data/events[at0001]/data/items[at1100]/value/magnitude
```

The codes [atnnnn] at each node of the archetype become the *archetype_node_ids* found in each node in the data.

Now consider an OBSERVATION instance (expressed here in dADL format), in which a history of two blood pressures has been recorded using this archetype:

```
<                              -- OBSERVATION - blood pressure measurement
       archetype_node_id = <[openEHR-EHR-OBSERVATION.bp_meas.v2]>
       name = <value = <“BP measurement”>>
       data = <                      -- HISTORY
          origin = <2005-12-03T09:22:00>
          events = <                       -- List <EVENT>
             [1] = <                             -- EVENT
                archetype_node_id = <[at0001]>
                name = <value = <“sitting”>>
                time = <2005-12-03T09:22:00>
                data = <                               -- ITEM_LIST
                   items = <                           -- List<ELEMENT>
                      [1] = <
                         name = <value = <“systolic”>>
                         archetype_node_id = <[at1100]>
                         value = <magnitude = <120.0> ...>
                      >
                      [2] = <....
                         name = <value = <“diastolic”>>
                         archetype_node_id = <[at1200]>
                         value = <magnitude = <80.0> ...>
                      >
                   >
                >
             >
             [2] = <                             -- EVENT
                archetype_node_id = <[at0001]>
                name = <value = <“standing”>>
                time = <2005-12-03T09:27:00>
                data = <                               -- ITEM_LIST
                   items = <                           -- List<ELEMENT>
                      [1] = <
                         name = <value = <“systolic”>>
                         archetype_node_id = <[at1100]>
                         value = <magnitude = <105.0> ...>
                      >
                      [2] = <
                         name = <value = <“diastolic”>>
                         archetype_node_id = <[at1200]>
                         value = <magnitude = <70.0> ...>
                      >
                   >
                >
             >
          >
>
```

[Note: in the above example, name values are shown as if they were all DV_TEXTs, whereas in reality in *open*EHR they more likely to be DV_CODED_TEXT instances; either is allowed by the archetype. This has been done to reduce the size of the example, and makes no difference to the paths shown below].

The archetype path mentioned above matches both systolic pressures in the recording. In many querying situations, this may be exactly what is desired. However, to uniquely match each of the systolic pressure nodes, paths need to be created that are based not only on the *archetype_node_id* but also on another attribute. In the case above, the *name* attribute provides uniqueness. Guaranteed unique paths to the systolic and diastolic pressures of the each event (sitting and standing measurements) are available using the following expressions (identical in Xpath):

```
/data/events[1]/data/items[1]/value/magnitude
/data/events[1]/data/items[2]/value/magnitude
/data/events[2]/data/items[1]/value/magnitude
/data/events[2]/data/items[2]/value/magnitude
```

More expressive unique paths based on archetype paths are also possible, as follows:

```
/data/events[at0001 AND
        name/value='sitting']/data/items[at1100]/value/magnitude
/data/events[at0001 AND
        name/value='sitting']/data/items[at1200]/value/magnitude
/data/events[at0001 AND
        name/value='standing']/data/items[at1100]/value/magnitude
/data/events[at0001 AND
        name/value='standing']/data/items[at1200]/value/magnitude
```

Each of these paths has an Xpath equivalent of the following form:

```
/data/events[@archetype_node_id = 'at0001' AND
        name/value='standing']/data/items[@archetype_node_id = 'at1200']
        /value/magnitude
```

As a general rule, one or more other attribute values in the runtime data will uniquely identify any node in *open*EHR data. To make construction of unique paths easier, the value of the *name* attribute (inherited from the LOCATABLE class), is *required* to be unique with respect to the name values of sibling nodes. This has two consequences as follows:

- a guaranteed unique path can always be constructed to any data item in *open*EHR data using a combination of *archetype_node_id* and *name* values (as shown in the example paths above);
- the *name* value may be systematically defined to be a copy of one or more other attribute values. For example, in an EVENT object, name could clearly be a string copy of the *time* attribute.

11.3 EHR URIs

To create a reference to a node in an EHR in the form of a URI (uniform resource identifier), two elements are needed: the path *within* a top-level structure, and a reference *to* a top-level structure. These are combined to form a URI in an "ehr" scheme-space, obeying the following syntax:

```
ehr://top_level_structure_locator/path_inside_top_level_structure
```

Under this scheme, any object in any *open*EHR EHR is addressable via a URI. The *open*EHR data type DV_EHR_URI is designed to carry URIs of this form, enabling URIs to be constructed for use within LINKs and elsewhere in the *open*EHR EHR. (URIs of course are only one method of addressing or querying data in the EHR. Other querying syntaxes and functional interfaces will be developed and used over time.)

11.3.1 Locating Top-Level Structures

The first part of an EHR URI needs to identify a top-level structure. As described above, a Version locator can be used to do this. However, this is not the only way: various logical queries can also be used, e.g. "get the latest version from a given Versioned object matching...". For reasons of efficiency, the top-level structure locator part of the URI is also likely to include the EHR id, and possibly the EHR system id, even though neither of these are strictly needed for identification. Thus, the first part of an EHR URI might include the following:

- EHR id;
- EHR system id, depending on whether the EHR id is globally unique or not;
- Either:
  - version time, i.e. time baseline for retrieving versions, defaults to "now";
  - identifier of particular Version container object, typically its *uid*;
- Or:
  - a Version identifier, i.e. {Version container object Uid; *version_tree_id*; *creating_system_id*}

For a number of reasons there is currently no standard syntax for encapsulating these parameters. Firstly, there is an issue to do with how EHRs will be identified in *open*EHR systems. Identifiers may be required to conform to local health jurisdiction requirements, or may not. If EHR identifiers are globally unique, or even nationally unique, then in theory the EHR system identifier can be dispensed with. However in a practical sense the identifier of the EHR system can only be dispensed with if there is a health information location service operating in the environment that can perform EHR id -> EHR system id mappings, in much the same way as the internet DNS converts logical domain names to IP addresses.

Another issue is whether it can be assumed that the version time baseline has already been established in some earlier call or service invocation, and that no version time information is needed.

Various syntax possibilities include:

- an Xpath-style syntax; this does not seem desirable as it implies hierarchical data containment structures that don't exist and is likely to be confused with the path part of the URI;
- a web-services inspired functional syntax;
- a database-inspired query syntax.

Currently, a fairly typical URI query style of syntax is used, as shown in the following examples.

- This path matches a `VERSIONED_COMPOSITION`:

```
ehr://rdh.health.gov.au?ehr=1234567&versioned_composition=87284370-2D4B-
4e3d-A3F3-F303D2F4F34B
```

- The following path matches the most recent `VERSION<COMPOSITION>` from a specified `VERSIONED_COMPOSITION`:

```
ehr://rdh.health.gov.au?ehr=1234567&versioned_composition=0892BF98-910D-
4df9-BF7E-F10D72C1C81A&latest_version
```

- The following path matches a `COMPOSITION` within `VERSION` 2 of a `VERSIONED_COMPOSITION`:

```
ehr://rdh.health.gov.au?ehr=1234567&version={F7C5C7B7-75DB-
4b39-9A1E-C0BA9BFDBDEC::rdh.health.gov.au::2}&data
```

In these paths, the following pseudo-identifiers are used:

- versioned_composition: to indicate an instance of VERSIONED_COMPOSITION;
- version_tree_id: the version identifier of the VERSION within the tree structure of the owning VERSIONED_OBJECT;
- latest_version: a pseudo-identifier used to indicate a VERSION instance being the result of the call VERSIONED_COMPOSITION.*latest_version*;

Implementors and users of the current release of *open*EHR are encouraged to experiment and/or propose improved solutions to the locator requirement described in this section.

12 Deployment

12.1 Correspondence to System Architectures

The previous section described the software package structure of the *open*EHR specifications. Here we describe how the package architecture can be applied to building real systems. The general architectural approach in any *open*EHR system can be considered as 5 layers (i.e. a “5-tier” architecture). The tiers are as follows.

1. **persistence**: data storage and retrieval.
2. **back-end services**: including EHR, demographics, terminology, archetypes, security, record location, and so on. In this layer, the separation of the different services is transparent, and each service has a coarse-grained service interface.
3. **virtual EHR**: this tier is the middleware, and consists of a coherent set of APIs to the various back-end services providing access to the relevant services, thereby allowing user access to the EHR; including EHR, demographics, security, terminology, and archetype services. It also contains an archetype- and template-enabled kernel, the component responsible for creating and processing archetype-enabled data. In this tier, the separation of back-end services is hidden, only the functionality is exposed. Other virtual clients are possible, consisting of APIs for other combinations of back-end services.
4. **application logic**: this tier consists of whatever logic is specific to an application, which might be a user application, or another service such as a query engine.
5. **presentation layer**: this layer consists of the graphical interface of the application, where applicable.

The same tiers can be used in large deployments, as shown in FIGURE 29, or simply as layers in single-machine applications.

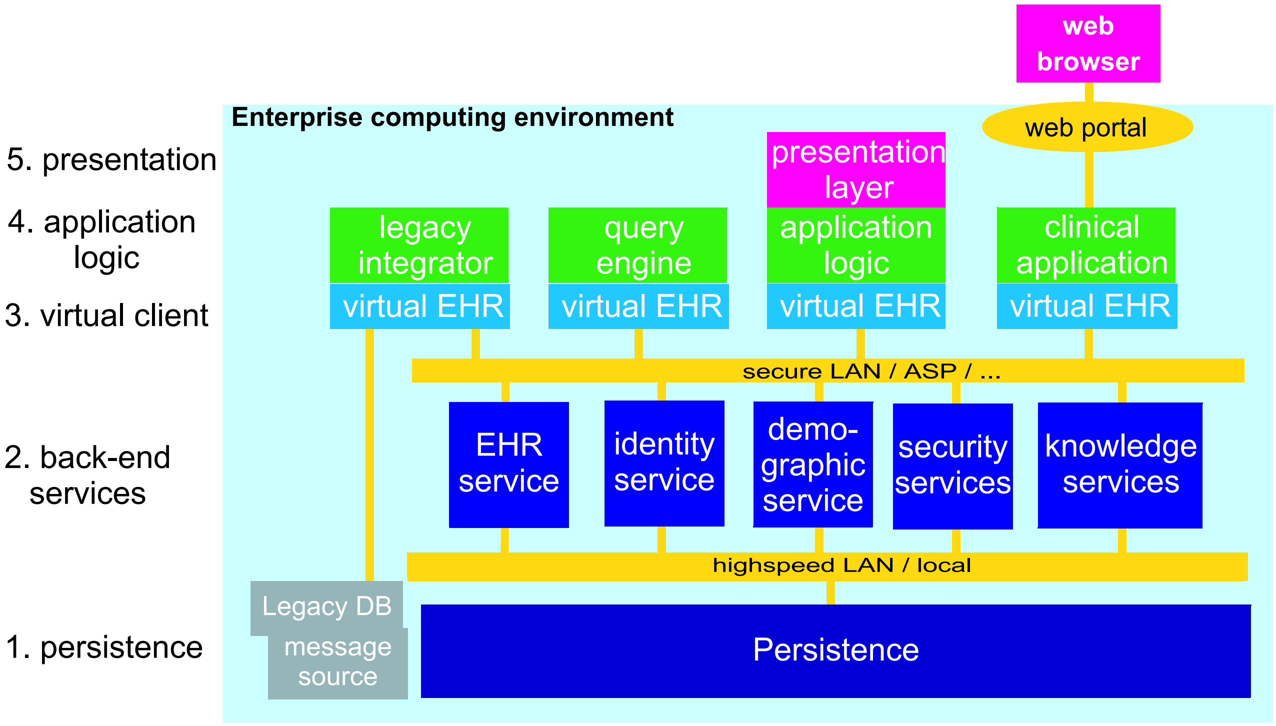


FIGURE 29 Basic Enterprise EHR System Architecture

FIGURE 30 illustrates an approximate mapping of major parts of the *open*EHR software architecture to the 5-tier scheme. Clearly where parts of the architecture are used will depend on various implementation choices; the mapping shown is therefore not definitive. Nevertheless, the principal use of parts of the architecture is likely to be similar in most systems, as follows:

- RM and AM: mainly used to construct an archetype- and template-processing kernel;
- RM `common.change_control` package: provides the logic for versioning in versioned services such as the EHR and demographics;
- SM: various service model packages define the exposed interfaces of major services;
- SM `virtual_ehr` package defines the API of the virtual EHR component;
- archetypes: archetypes might be assumed directly in some applications, e.g. a specialist peri-natal package might be partly based on a family of archetypes for this specialisation;
- templates: both archetypes and templates will be used in the presentation layer of applications. Some will base the GUI code on them, while others will have either tool-generate code, or dynamically generate forms based on particular templates and archetypes.

In the future, an abstract persistence API and optimised persistence models (transformations of the existing RM models) are likely to be published by *open*EHR in order to help with the implementation of databases.

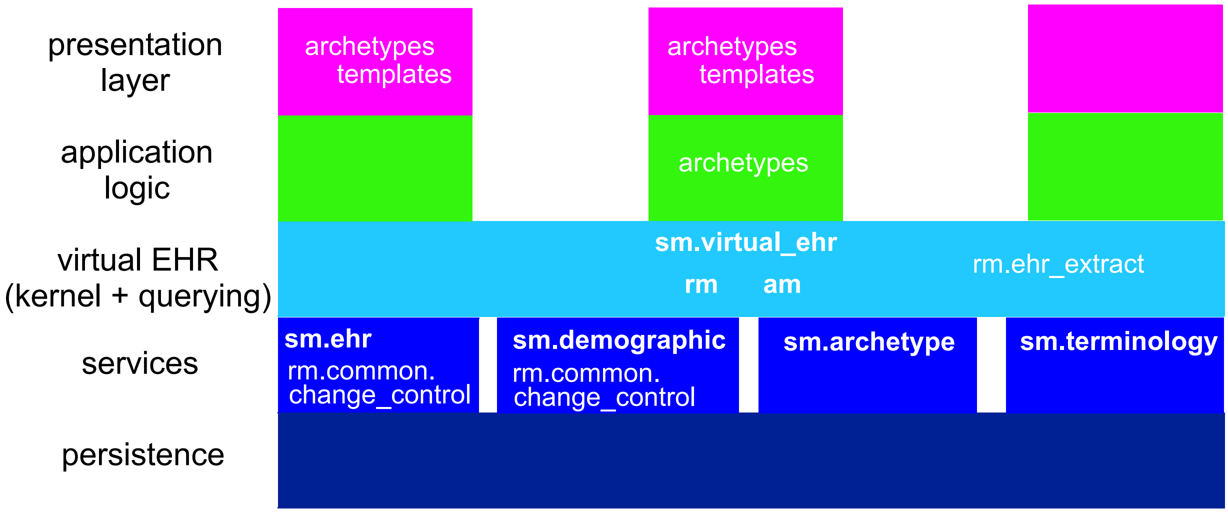


FIGURE 30 Mapping of software architecture to systems

13 Relationship to Standards

The *open*EHR specifications make use of available standards where relevant, and as far as possible in a compatible way. However, for the many standards have never been validated in their published form (i.e. the form published is not tested in implementations, and may contain errors), *open*EHR makes adjustments so as to ensure quality and coherence of the *open*EHR models. In general, "using" a standard in *open*EHR may mean defining a set of classes which map it into the *open*EHR type system, or wrap it or express it in some other compatible way, allowing developers to build completely coherent *open*EHR systems, while retaining compliance or compatibility with standards. The standards relevant to *open*EHR fall into a number of categories as follows.

Standards by which *open*EHR can be evaluated

These standards define high-level requirements or compliance criteria which can be used to provide a means of normative comparison of *open*EHR with other related specifications or systems:

- **ISO/TR 20514**. Health informatics — Electronic health record — Definition, scope, and context. ISO TC 215/WG 1.
- **ISO/TS 18308**. Technical Specification for Requirements for an EHR Architecture. ISO TC 215/WG1.

Standards which have influenced the design of *open*EHR specifications

The following standards have influenced the design of the *open*EHR specifications:

- **OMG HDTF** Standards - general design
- **CEN EN 13606:2006**: Electronic Health Record Communication
- **CEN HISA 12967-3**: Health Informatics Service Architecture - Computational viewpoint

Standards which have influenced the design of *open*EHR archetypes

The following standards are mainly domain-level models of clinical practice or concepts, and are being used to design *open*EHR archetypes and templates.

- **CEN HISA 12967-2**: Health Informatics Service Architecture - Information viewpoint
- **CEN ENV 13940**: Continuity of Care.

Standards which are used "inside" *open*EHR

The following standards are used or referenced at a fine-grained level in *open*EHR:

- **ISO 8601**: Syntax for expressing dates and times (used in *open*EHR Quantity package)
- **ISO 11404**: General Purpose Datatypes (mapped to in *open*EHR `assumed_types` package in Support Information Model)
- **HL7 UCUM**: Unified Coding for Units of Measure (used by *open*EHR Data types)
- **HL7v3 GTS**: General Timing Specification syntax (used by *open*EHR Data types).
- some HL7v3 domain vocabularies are mapped to from the *open*EHR terminology.
- **IETF RFC 2440** - openPGP.

Standards which require a conversion gateway

The following standards are in use and require data conversion for use with *open*EHR:

- **CEN EN 13606:2005**: Electronic Health Record Communication - near-direct conversion possible, as *open*EHR and CEN EN 13606 are actively maintained to be compatible.

- **HL7v3 CDA**: Clinical Document Architecture (CDA) release 2.0 - fairly close conversion may be possible.
- **HL7v3 messages**. Quality of conversion currently unknown due to flux in HL7v3 messaging specifications.
- **HL7v2 messages**. Experience in Australia indicates that importing of HL7v2 message information is relatively easy. Export from *open*EHR may also be possible.

Generic Technology Standards

The following standards are used or referenced in *open*EHR:

- ISO RM/ODP
- OMG UML 2.0
- W3C XML schema 1.0
- W3C Xpath 1.0

14 Implementation Technology Specifications

14.1 Overview

ITSs are created by the application of transformation rules from the “full-strength” semantics of the abstract models to equivalents in a particular technology. Transformation rules usually include mappings of:

- names of classes and attributes;
- property and function signature mapping;
- mapping of basic types e.g. strings, numerics;
- how to handle multiple inheritance;
- how to handle generic (template) types;
- how to handle covariant and contravariant redefinition semantics;
- the choice of mapping properties with signature *xxxx:T* (i.e. properties with no arguments) to stored attributes (*xxxx:T*) or functions (*xxxx():T*);
- how to express preconditions, postconditions and class invariants;
- mappings between assumed types such as `List<>`, `Set<>` and inbuilt types.

ITSs are being developed for a number of major implementation technologies, as summarised below. Implementors should always look for an ITS for the technology in question before proceeding. If none exists, it will need to be defined. A methodology to do this is being developed.

FIGURE 31 illustrates the implementation technology specification space. Each specification documents the mapping from the standard object-oriented semantics used in the *open*EHR abstract models, and also provides an expression of each of the abstract models in the ITS formalism.

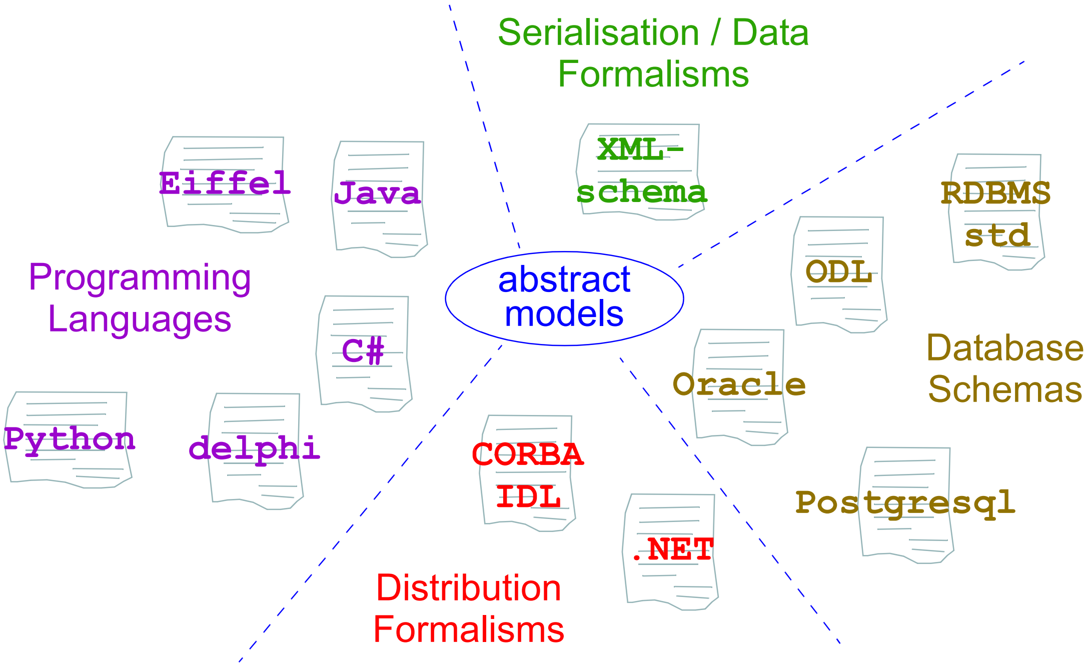


FIGURE 31 Implementation Technologies

A References

1 Beale T. *Archetypes: Constraint-based Domain Models for Future-proof Information Systems*. See http://www.deepthought.com.au/it/archetypes.html.

2 Beale T. *Archetypes: Constraint-based Domain Models for Future-proof Information Systems*. OOPSLA 2002 workshop on behavioural semantics. Available at http://www.deepthought.com.au/XXX.

3 ISO:IEC: Information Technology. *Open Distributed Processing, Reference Model: Part 2:Foundations*.

4 Maier M. *Architecting Principles for Systems-of-Systems*. Technical Report, University of Alabama in Huntsville. 2000. Available at http://www.infoed.com/Open/PAPERS/systems.htm

5 Rector A L, Nowlan W A, Kay S. *Foundations for an Electronic Medical Record*. The IMIA Yearbook of Medical Informatics 1992 (Eds. van Bemmel J, McRay A). Stuttgart Schattauer 1994.

6 Schloeffel P. (Editor). Requirements for an Electronic Health Record Reference Architecture. International Standards Organisation, Australia; Feb 2002; ISO TC 215/SC N; ISO/WD 18308.

7 CORBAmed document: *Person Identification Service*. (March 1999). (Authors?)

8 CORBAmed document: *Lexicon Query Service*. (March 1999). (Authors?)

END OF DOCUMENT