openEHR Architecture

Architecture Overview

Editors:{T Beale, S Heard}[1], {D Kalra, D Lloyd}[2]

Revision: 1.1

Pages: 47

1. Ocean Informatics Australia
2. Centre for Health Informatics and Multi-professional Education, University College London

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. **CR-000200**. Correct package names in RM diagram. Added configuration management and versioning material from Common IM. Added section on ontological landscape. Added section on aims. Added section on systems architectures.	T Beale D Lloyd T Beale	06 Mar 2006
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.

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

1.2 Related Documents

Prerequisite documents for reading this document include:

- The *open*EHR Roadmap document
- The *open*EHR Modelling Guide

Other documents describing related models, include:

- The *open*EHR Information Model documents
- The *open*EHR Archetype Model documents

1.3 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.

1.4 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 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).

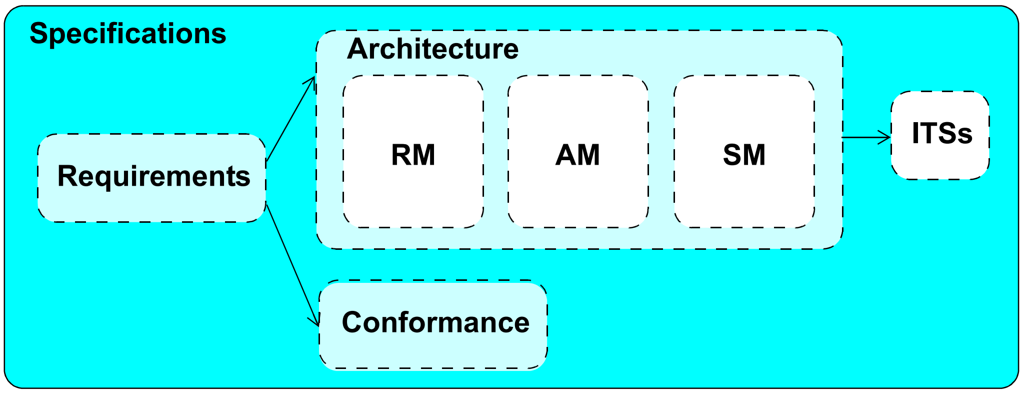

FIGURE 1 *open*EHR Specification project

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.

All of 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 what is done in any implementation domain. 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 abstract 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 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

Types of System

This section provides a brief overview of the aims of the *open*EHR specifications, as an aid to understanding the remainder of the document. The architecture of *open*EHR is designed to support the construction of a number of types of system. One of the most important could be characterised as a distributed, patient-centred, life-long, shared care health record, illustrated in FIGURE 2.

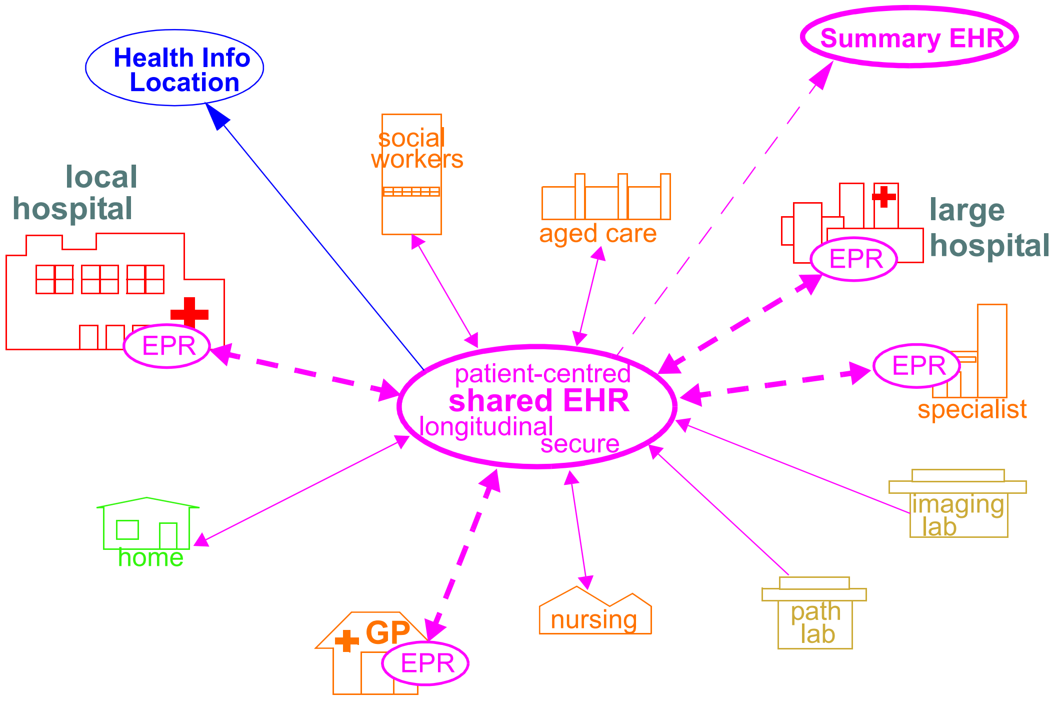

FIGURE 2 Community Shared-care Context

In this form, the *open*EHR services are added to the existing IT infrastructure 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:

- 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.

Requirements

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. Among the global requirements of EHRs and EHR systems supported by *open*EHR are the following:

- a life-long EHR;

- prioritises the patient / clinician interaction;
- medico-legal faithfulness, traceability, audit-trailing;
- technology & data format independent;
- facilitate sharing of EHRs via interoperability at data and knowledge levels;
- suitable for both primary & acute care;
- integrates with any/multiple terminologies;
- supports all natural languages, as well as translations between languages in the record;
- support for patient privacy, including anonymous EHRs;
- support for clinical data structures: lists, tables, time-series, including point and interval events;
- support for all aspects of pathology data, including normal ranges, alternative systems of units etc;
- highly maintainable and flexible software;
- compatibility with CEN 13606, Corbamed, and messaging systems;
- support semi-automated and automated distributed workflows;
- supports secondary uses: education, research, population medicine.

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.

1. see http://www.openehr.org/downloads/ISOEHRRequirements.zip
2. see http://svn.openehr.org/specification/TRUNK/publishing/requirements/iso18308_conformance.pdf

4 Design Principles

The *open*EHR approach to modelling information, services and domain knowledge is based on a number of design principles, described below. All of these principles lead to a separation of the models of the openEHR 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 separation in any system of models is ontological, i.e. in the semantic dimension. 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 terminology will describe 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 3 illustrates these distinctions, and indicates what parts are built directly into software and databases.

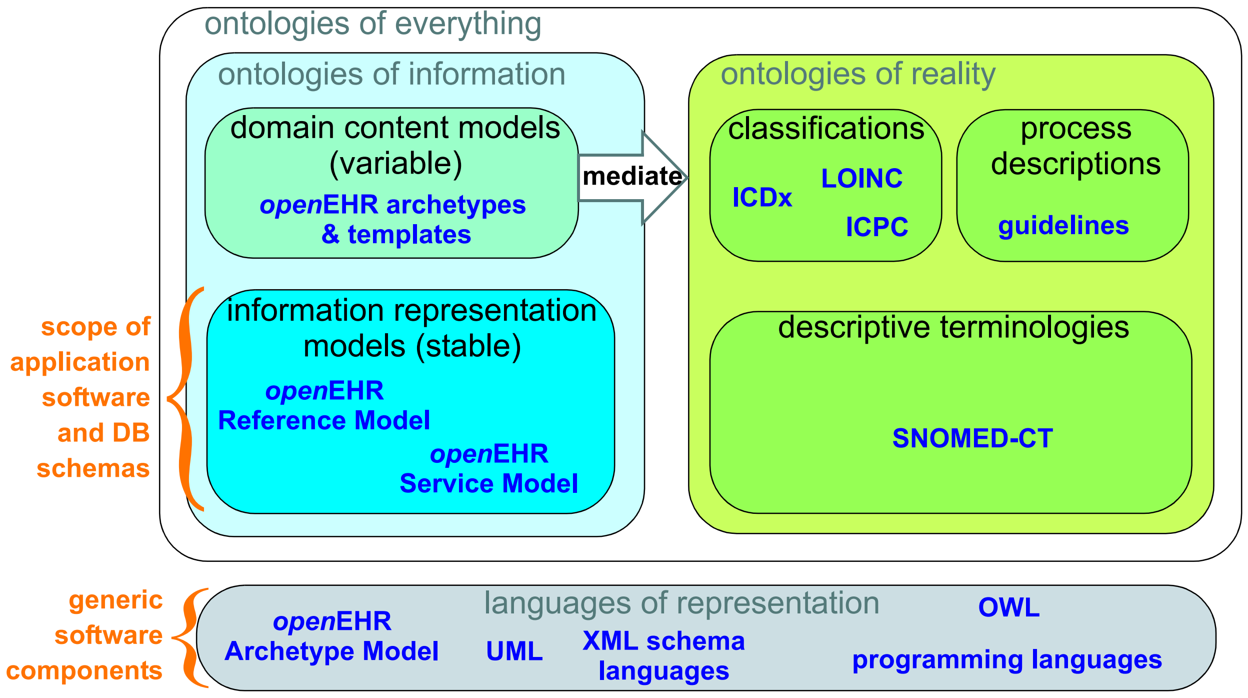
ontologies of everything
ontologies of information
domain content models (variable)
openEHR archetypes & templates
mediate
information representation models (stable)
openEHR Reference Model
openEHR Service Model
ontologies of reality
classifications
ICDx
LOINC
ICPC
process descriptions
guidelines
descriptive terminologies
SNOMED-CT
scope of application software and DB schemas
generic software components
languages of representation
openEHR Archetype Model
UML
XML schema languages
OWL
programming languages

FIGURE 3 The Ontological Landscape

This figure shows a primary separation between "ontologies of information" i.e. models of information content, from "ontologies of reality" i.e. descriptions and classifications of real phenomena. These two categories have to be separated because the 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.

A secondary ontological separation within the "information" side is shown between information models and domain content models. This separation is not generally well understood, and historically, the entirety of informational semantics has been hard-wired into the software and databases, leading to unmaintainable systems.

By clearly separating the three elements - 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 3). As a consequence, systems have the possibility of being far smaller and more maintainable. 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 openEHR engenders new relationships between information and models, as shown in FIGURE 4. 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).

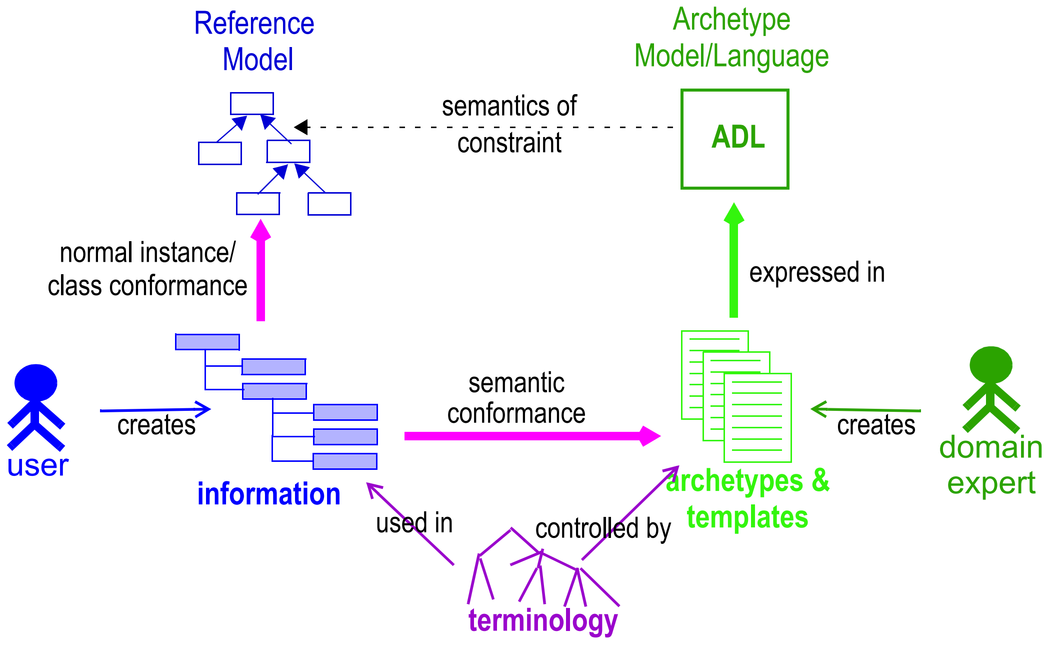

FIGURE 4 Archetype Meta-architecture

The details of how archetypes and templates work in *open*EHR are described in Archetyping on page 33.

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 ISO Reference Model for Open Distibuted Processing (RM-ODP) [3]. 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 5 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 create coherent structural definitions in the informational and computational viewpoints that 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], which are as follows:

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 44), 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 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

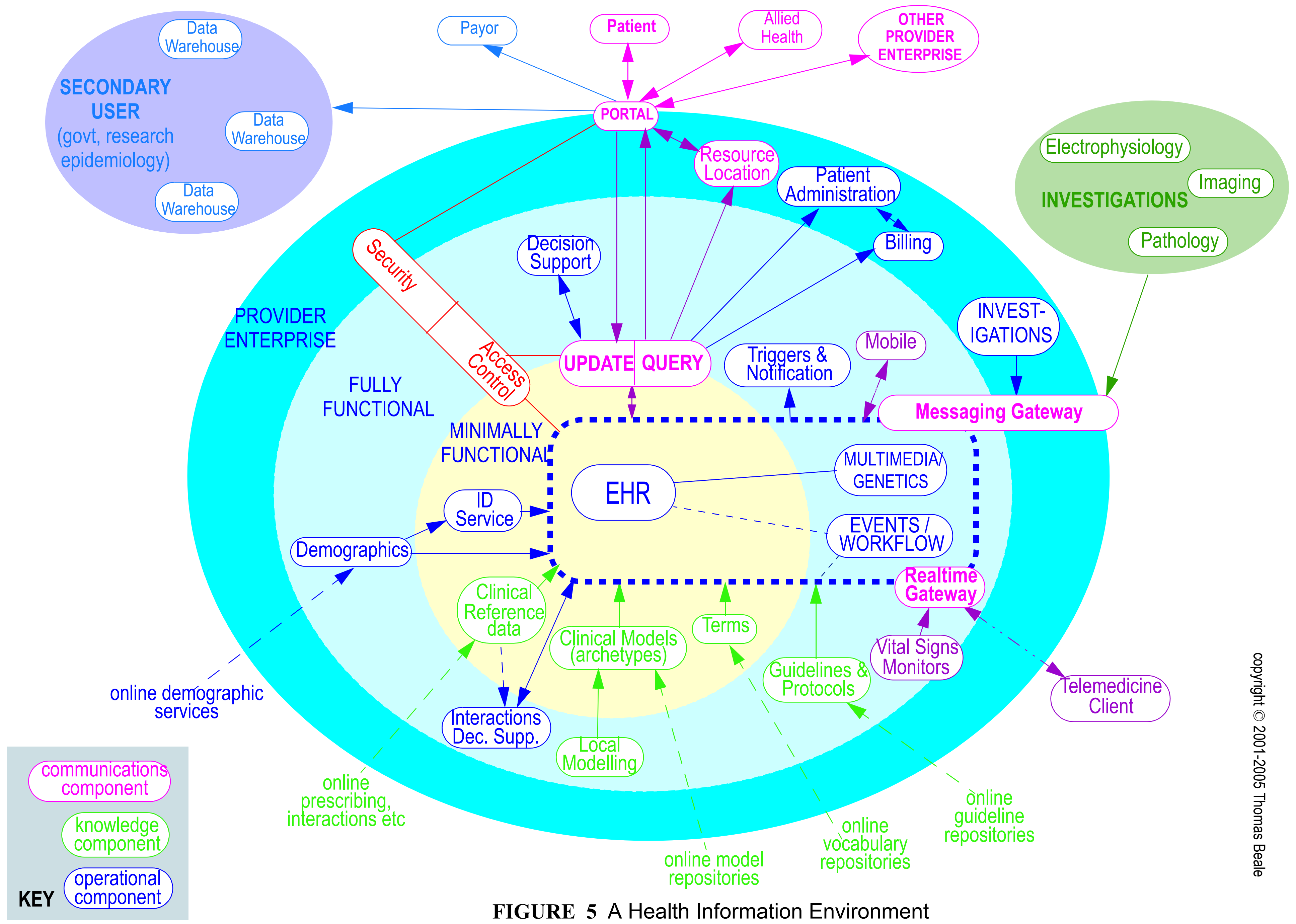

FIGURE 5 A Health Information Environment

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 8 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 is usually represented in UML as further packages. In some implementation technologies (e.g. java), the `org.openehr` namespace may actually be used within program texts.

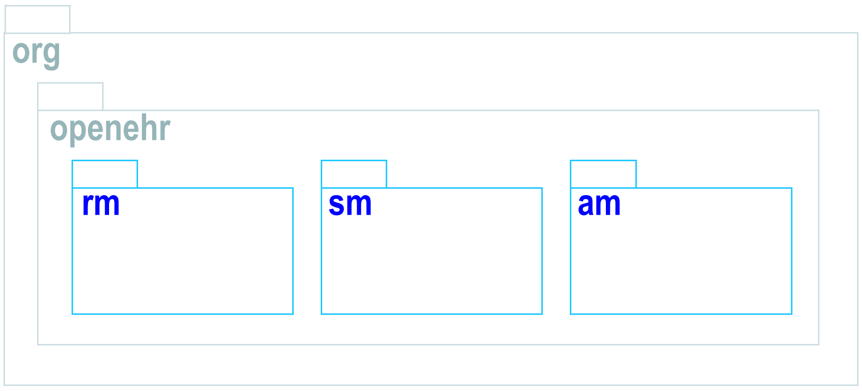

FIGURE 6 Global Package Structure of *open*EHR

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 within the *open*EHR framework. 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 7 illustrates the relationships between the outer packages and their main constituents. The common type platform is shown at the bottom.

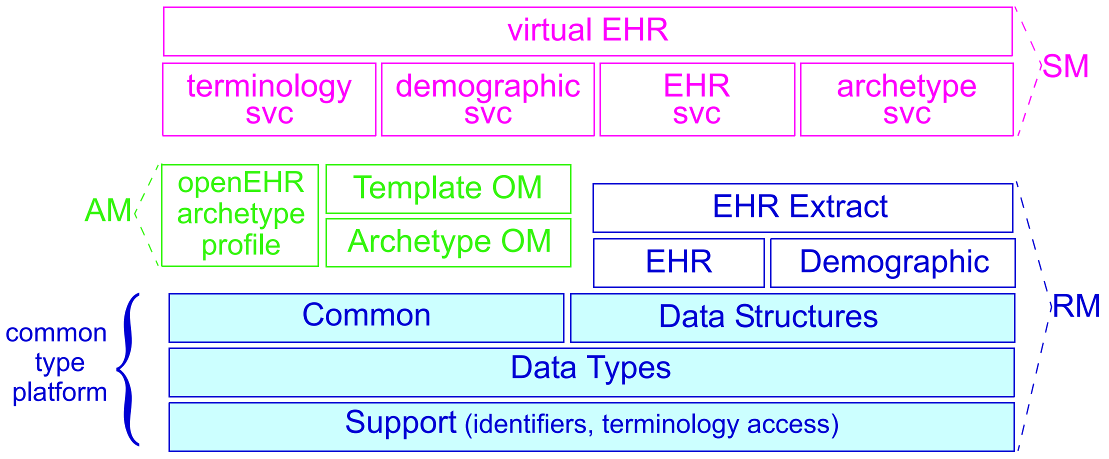

FIGURE 7 Computing Platform view of packages

5.2 Reference Model (RM)

Within the any given namespace, each package defines a local context for definition of classes. FIGURE 8 illustrates the package structure in the RM namespace. 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 8 corresponds to one *open*EHR specification document[1], documenting an "information model" (IM). Where packages are nested, the inner packages cannot exist outside of their parent package. 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 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 many reference models, generic data structures are available 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.

1. with the exception of the EHR and Composition packages, which are both described in the EHR Reference Model document; this may change in the future.

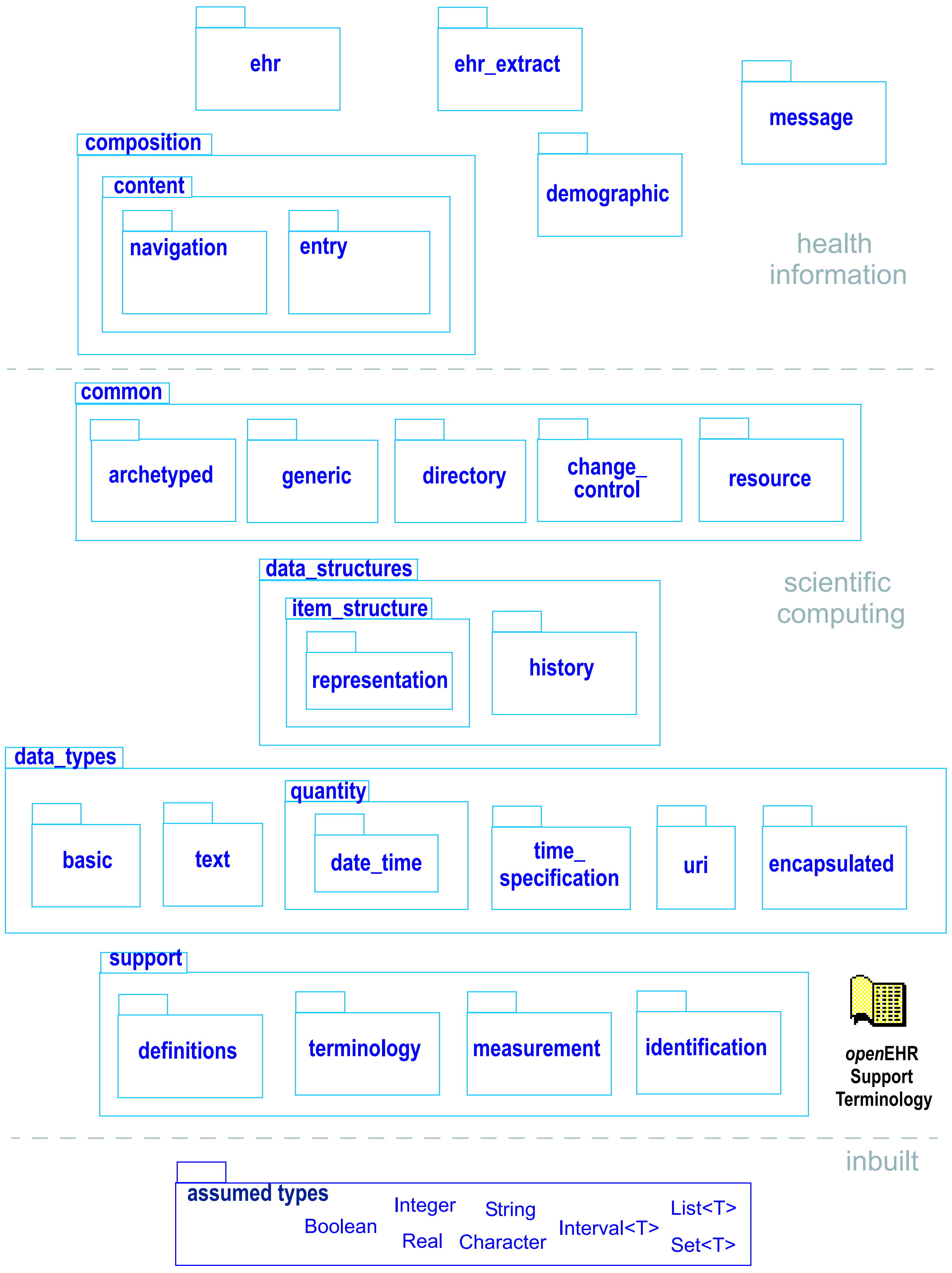

FIGURE 8 Structure of org.openehr.rm package

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.

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 semantic concepts are used in common by various models. 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 last group of concepts consists of a formal model of change management which applies to any service that needs to be able to supply previous states of its information, in particular the demographic and EHR services.

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.

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 9.

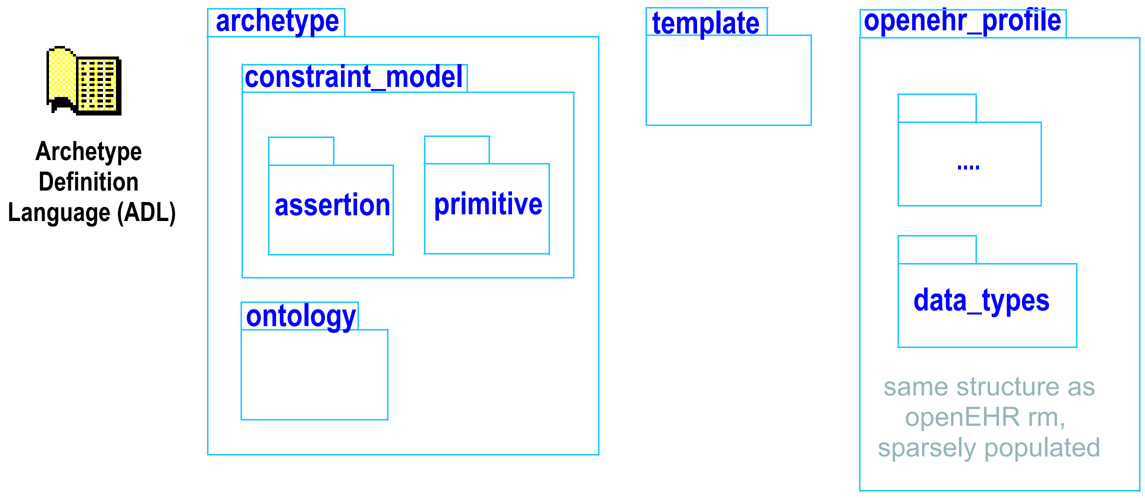

FIGURE 9 Structure of the org.openehr.am package

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 10. The set of services actually included will undoubtedly evolve over time, so this diagram should not be seen as definitive.

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. The finest object that can be requested or committed via the EHR service is a single Composition, or the EHR Directory structure.

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.

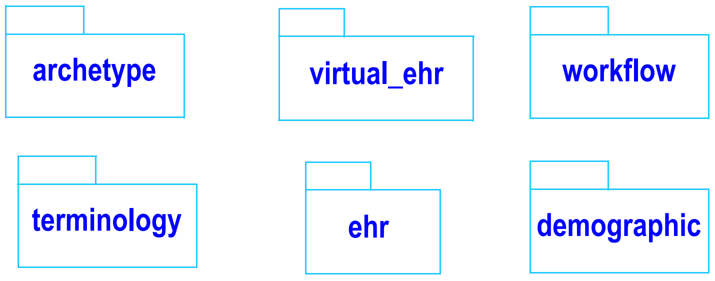

FIGURE 10 Structure of the org.openehr.sm package

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 Deployment

6.1 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 11, or simply as layers in single-machine applications.

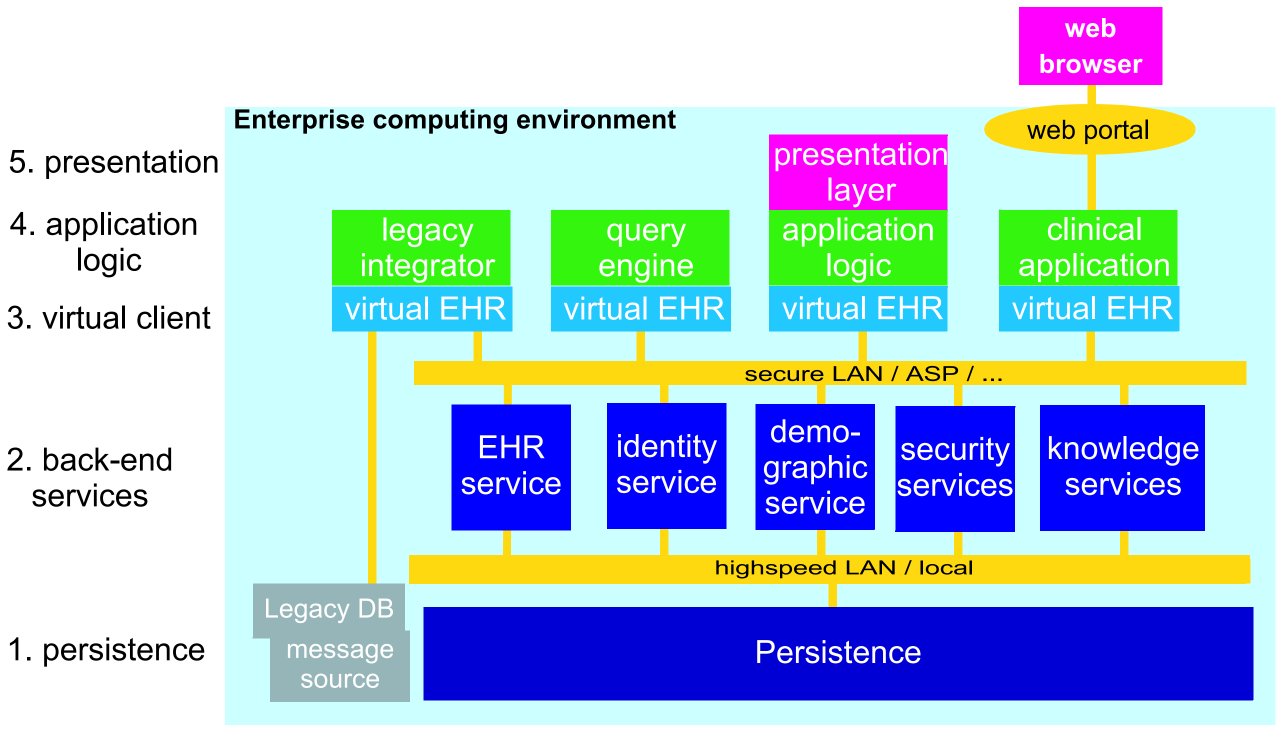

FIGURE 11 Basic Enterprise EHR System Architecture

FIGURE 12 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.

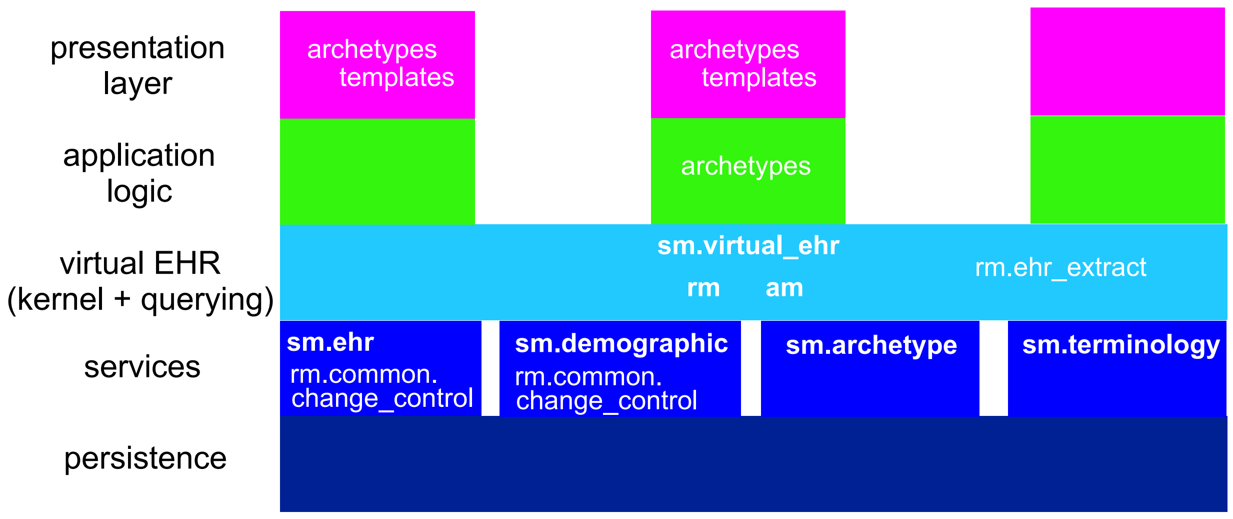

FIGURE 12 Mapping of software architecture to systems

6.2 Top-level Information Structures

The *open*EHR information models define various informational artifacts 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 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.

7 Versioning

7.1 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 13.

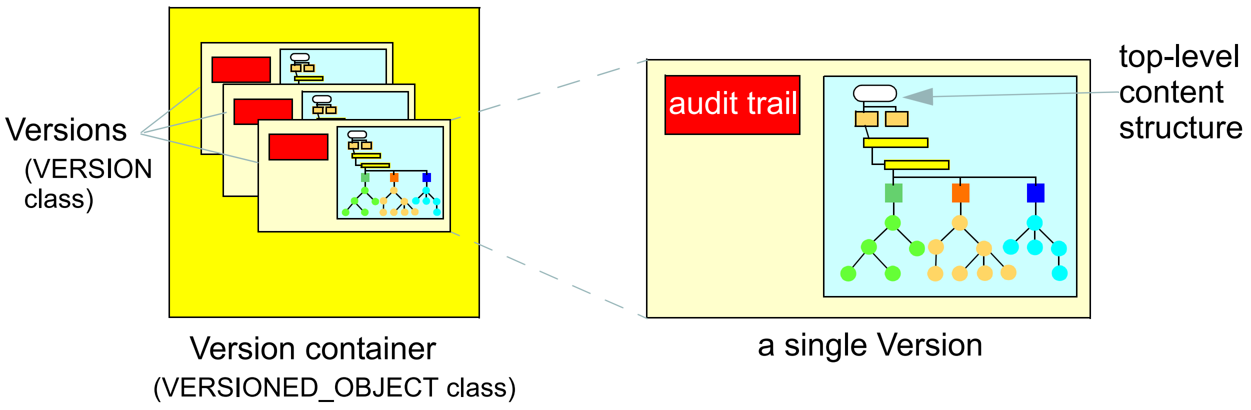

FIGURE 13 Version-control structures

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

7.2 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.

7.2.1 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 14.

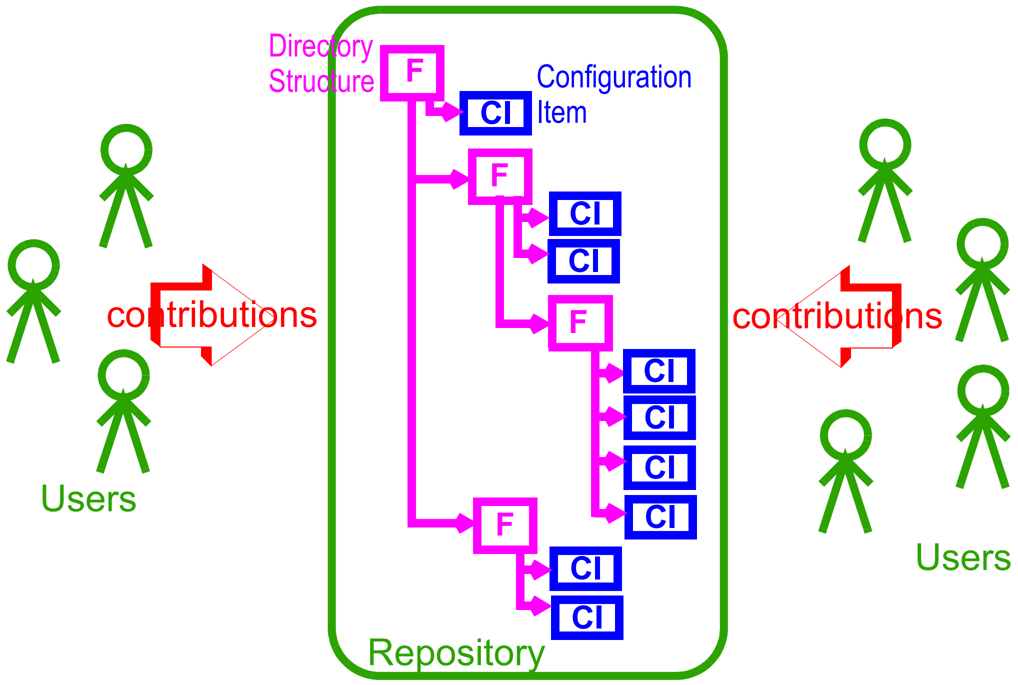

FIGURE 14 General Structure of a Controlled Repository

7.2.2 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.

7.3 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 15 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).

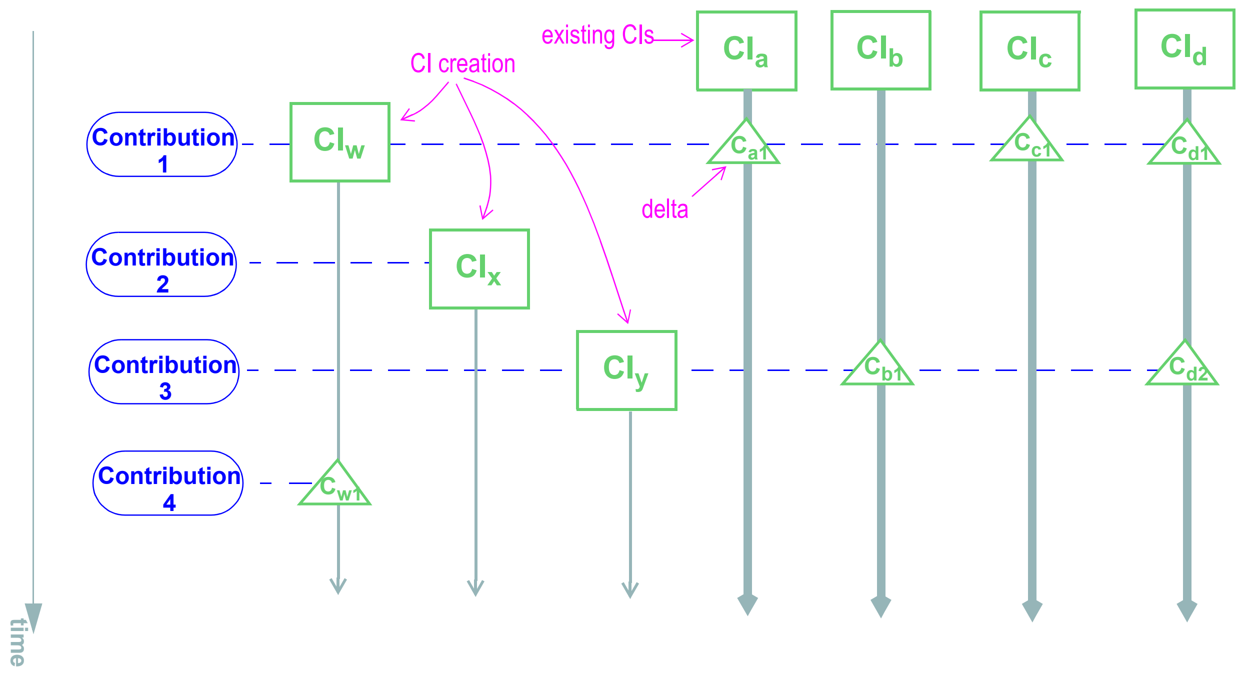

FIGURE 15 Contributions to the Repository (delta form)

One nuance which should be pointed out is that in FIGURE 15, 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 15; 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 16. 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.

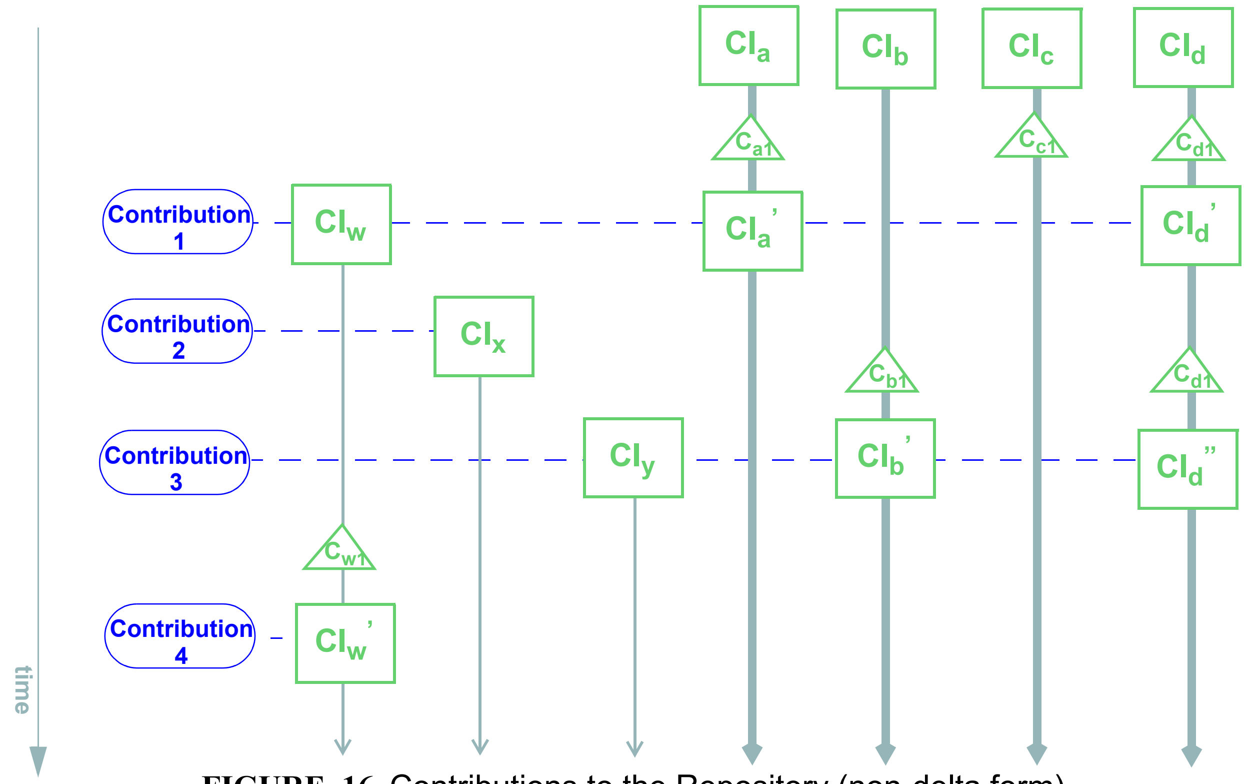

FIGURE 16 Contributions to the Repository (non-delta form)

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 16.

7.3.1 General Model of a Change-controlled Repository

FIGURE 17 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 17 therefore provides a guide to the definition of actual controlled repositories, such as an EHR, rather than a formal specification for them.

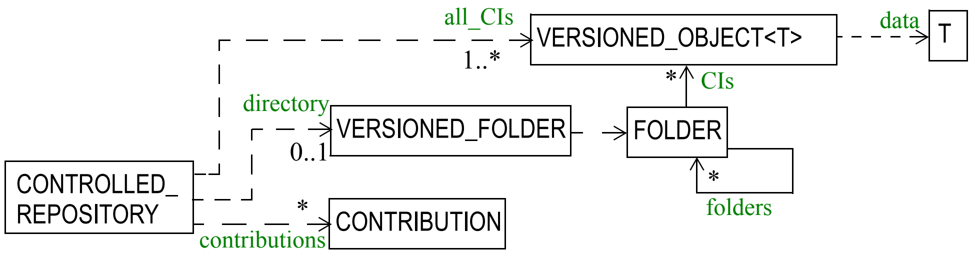

FIGURE 17 Abstract Model of Change-controlled Repository

7.4 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.

8 Identification

8.1 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 18. Here we see two container objects with OBJECT_IDs (since OBJECT_ID is an abstract type, the actual type will be another XXX_ID class), and various OBJECT_REFs as references.

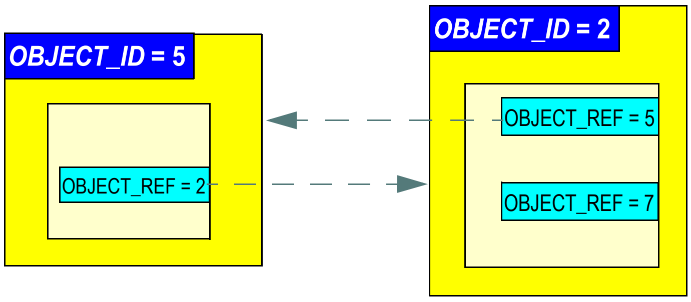

FIGURE 18 XXX_IDs and XXX_REFs

8.2 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_OBJECTs (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 19 summarises how various types of OBJECT_ID and OBJECT_REF are used to identify objects, and to reference them from the outside, respectively.

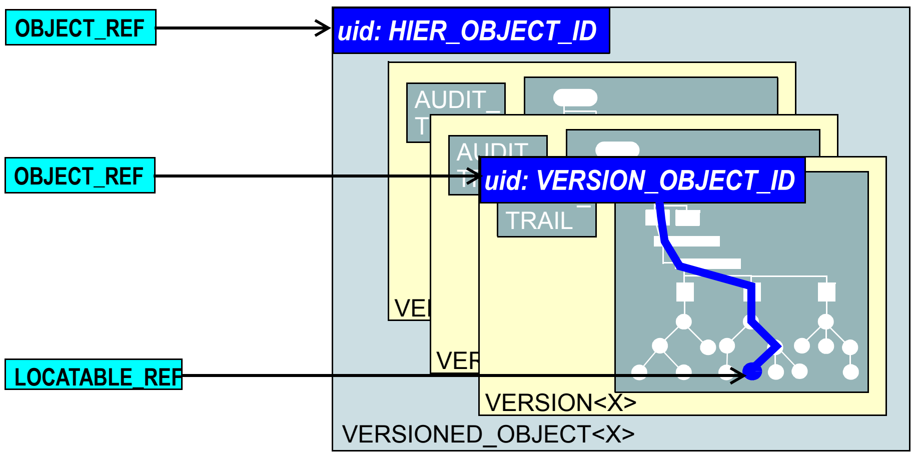

FIGURE 19 How to reference various levels of object

9 Archetyping

9.1 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.

9.2 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 20 illustrates the application of archetypes and templates to data.

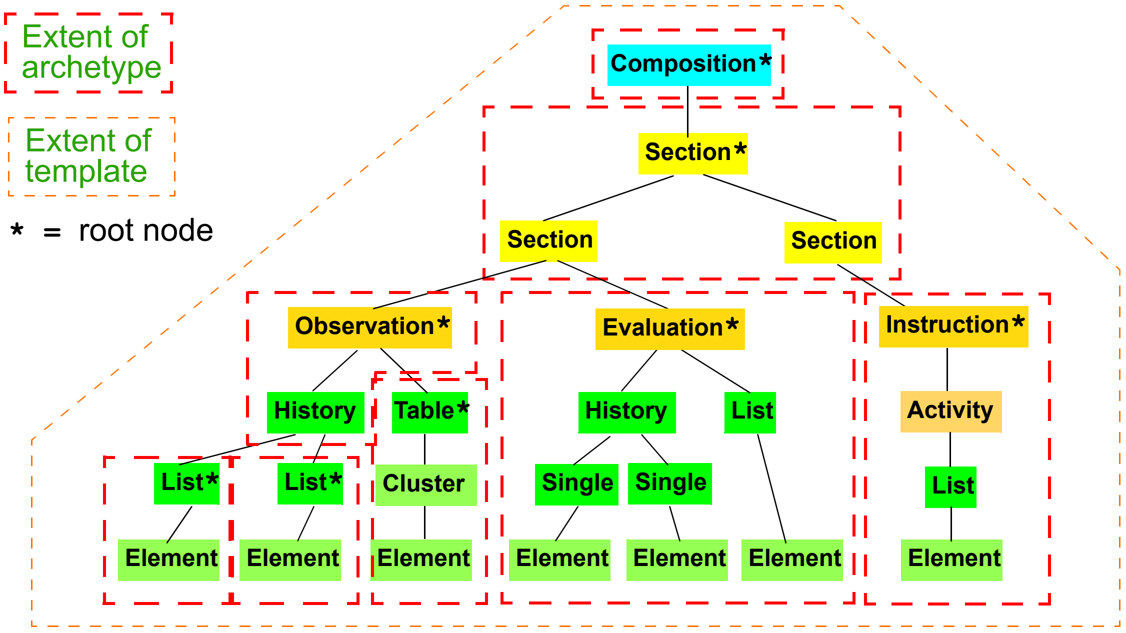

FIGURE 20 How Archetypes apply to Data

9.3 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 20 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.

9.4 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 details under Paths and Locators on page 36.

10 Paths and Locators

10.1 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.

10.2 Paths

10.2.1 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.

10.2.2 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"]
```

10.2.3 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 20. 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.

10.2.4 Runtime Paths and Uniqueness

Archetype paths are not guaranteed to be unique in data. However it will sometimes be 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 {...}
                        data matches {
                            ITEM_LIST matches {          -- systemic arterial BP
                                count matches {2..*}
                                items matches {
                                    ELEMENT[at1100] matches {-- systolic BP
                                        name matches {...}
                                        value matches {magnitude matches {...}}
                                    }
                                    ELEMENT[at1200] matches {-- diastolic BP
                                        name 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:

```
<                                         -- blood pressure measurement
        archetype_node_id = <[openEHR-EHR-OBSERVATION.bp_meas.v2]>
        name = <"xxx">
        data = <                                 -- HISTORY
           events = <                                  -- List <EVENT>
              [1] = <                                        -- EVENT
                 archetype_node_id = <[at0001]>
                 name = <"sitting">
                 data = <                                    -- ITEM_LIST
                    items = <                                -- List<ELEMENT>
                       ["systolic"] = <
                          archetype_node_id = <[at1100]>
                          value = <magnitude = <120.0> ...>
                       >
                       ["diastolic"] = <....
                          archetype_node_id = <[at1200]>
                          value = <magnitude = <80.0> ...>
                       >
                    >
                 >
              >
              [2] = <                                        -- EVENT
                 archetype_node_id = <[at0001]>
                 name = <"standing">
                 data = <                                    -- ITEM_LIST
                    items = <                                -- List<ELEMENT>
                       ["systolic"] = <
                          archetype_node_id = <[at1100]>
                          value = <magnitude = <105.0> ...>
                       >
                       ["diastolic"] = <....
                          archetype_node_id = <[at1200]>
                          value = <magnitude = <70.0> ...>
                       >
                    >
                 >
              >
           >
>
```

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. Unique paths to the :

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

10.3 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.)

10.3.1 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.

11 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. The following ones have been used for this purpose so far:

- ISO TC 251 TS 18308 - Technical Specification for Requirements for an EHR Architecture.

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.

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

12 Implementation Technology Specifications

12.1 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 21 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.

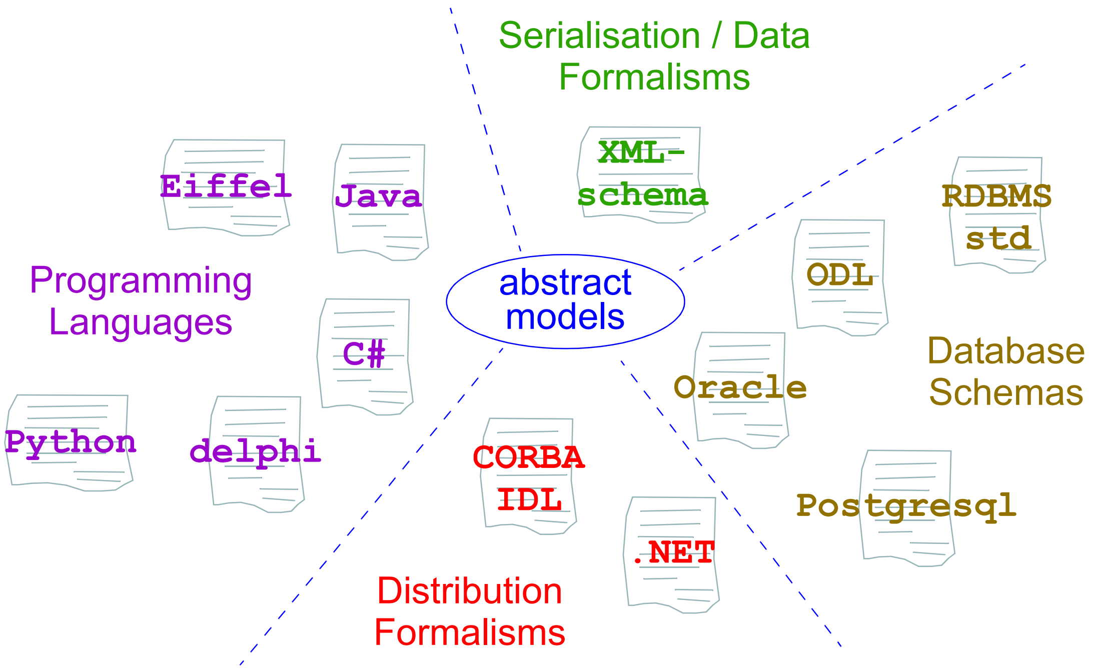

FIGURE 21 Implementation Technologies

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.deep-thought.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