



Object Databases and Object Persistence for openEHR

By:

Travis Muirhead

School of:

Computer and Information Science

Honours Thesis for:

Bachelor of Information Technology

(Advanced Computer and Information Science) (Honours)

Honours supervisors:

Name	Role	Association
Jan Stanek	Supervisor	UniSA
Heath Frankel	Associate Supervisor	Ocean Informatics
Chunlan Ma	Associate Supervisor	Ocean Informatics

Contents

Contents.....	i
List of Figures	iv
List of Tables.....	v
Acronyms	vii
Summary	viii
Declaration	ix
1 Introduction	10
1.1 Background and Motivation.....	10
1.2 Research Questions	13
2 openEHR Architecture	14
2.1 Modelling and Design foundations	14
2.2 Package Overview.....	16
2.3 Archetypes and Templates	16
2.4 Structure of the EHR and Compositions.....	19
2.5 Paths, Locators and Querying	21
3 Database Models	22
3.1 Relational Databases	22
3.2 XML enabled Relational Databases	24
3.3 Object-Oriented Databases	28
4 OODB Products and Technologies	30
4.1 dB4objects	31
4.1.1 Storing Objects using C#.....	31
4.1.2 Retrieving Objects using C#.....	32
4.1.3 Storage Capacity.....	33
4.1.4 Maintenance.....	34
4.1.5 Concurrency	34
4.1.6 Security.....	35
4.1.7 Distribution.....	35
4.1.8 Fault Tolerance and Availability	36

4.1.9	Support.....	36
4.1.10	Opportunities	36
4.2	Intersystems Caché	36
4.2.1	Creating Classes in Caché.....	37
4.2.2	Accessing the database from C#	37
4.2.3	Storing Objects in C#	37
4.2.4	Retrieving Objects in C#	38
4.2.5	Storage Capacity.....	38
4.2.6	Maintenance.....	38
4.2.7	Concurrency	39
4.2.8	Security and Encryption.....	39
4.2.9	Distribution.....	39
4.2.10	Fault Tolerance and Availability	40
4.2.11	Support.....	41
4.2.12	Opportunities	41
4.3	Objectivity/DB.....	41
4.3.1	Storing Objects in C#	41
4.3.2	Retrieving Objects in C#	43
4.3.3	Storage Capacity.....	43
4.3.4	Maintenance.....	44
4.3.5	Concurrency	44
4.3.6	Security and Encryption.....	45
4.3.7	Distribution.....	45
4.3.8	Fault Tolerance and Availability	46
4.3.9	Support.....	46
4.3.10	Opportunities	47
5	Preliminary Evaluation.....	48
5.1	Testing Environment.....	48
5.2	Measurement Toolkit	51
5.3	Object Model for Initial Evaluation.....	51
5.4	Implementation strategies	52
5.4.1	Db4o implementation	52
5.4.2	Intersystem's Caché implementation.....	52

5.4.3	Summary of Test Configurations	54
5.4.4	Bulk Insertion Time	55
5.4.5	Insertion at Fixed Intervals.....	56
5.4.6	Find different sized nodes.....	57
5.4.7	Find Single Node.....	57
5.4.8	Find Group	58
5.5	Preliminary Evaluation Summary.....	60
6	Final Evaluation	61
6.1	Prototype and Implementation Considerations.....	61
6.1.1	Persistence Layer Requirements and Use Cases.....	63
6.1.2	Global, Object Reference and Query Techniques	64
6.1.3	Implementation Issues.....	70
6.2	Test Data.....	71
6.3	Test Environment.....	72
6.4	Test Scenarios.....	74
6.5	Results and Comparison	75
6.5.1	Disk Space	75
6.5.2	Insertion.....	76
6.5.3	Find and Retrieve a COMPOSITION's Meta-data	78
6.5.4	Find and Retrieve a COMPOSITION by a unique identifier.....	78
6.5.5	Find and Retrieve a COMPOSITION based on its corresponding archetype.....	80
6.5.6	Find and Retrieve a CONTENT_ITEM based on the archetype and archetype node ..	81
7	Discussion	84
8	Conclusion.....	86
9	References	88
	Appendix A: Performance Measurement Toolkit	94
	Appendix B: Code used to manage globals and use cases	96
	Appendix C: Issues with Caché .NET Managed Provider	98
	Appendix D: Code fragments from the code generator	101

List of Figures

FIGURE 1	A Two-Level Modelling Paradigm.....	14
FIGURE 2	A Single-Level Modelling Paradigm.....	15
FIGURE 3	<i>openEHR</i> package structure	16
FIGURE 4	Archetype Software Meta-Architecture.....	17
FIGURE 5	Partial node map of an archetype for laboratory results	18
FIGURE 6	High-Level Structure of the <i>openEHR</i> EHR	19
FIGURE 7	Elements of an <i>openEHR</i> Composition.....	20
FIGURE 8	Partial view of the entry package, showing the subclasses of CARE_ENTRY	21
FIGURE 9	Comparison of join operations in an RDBMS to references in an OODBMS	23
FIGURE 10	SQL Server 2005 XML architecture overview	27
FIGURE 11	Persisting objects in db4o with C#	31
FIGURE 12	A typical AQL query	32
FIGURE 13	Retrieving objects from db4o with C# and SODA queries.....	33
FIGURE 14	Saving a Caché proxy object in C#.....	38
FIGURE 15	Storing a ooObj extended C# object to a default Objectivity cluster.....	42
FIGURE 16	Results from 'HD Tune' Benchmark for the Western Digital Hard Drive	50
FIGURE 17	Linear Recursive Structure used for Preliminary Testing.....	52
FIGURE 18	Preliminary Evaluation: Bulk Insertion Time.....	55
FIGURE 19	Preliminary Evaluation: Insertion at Fixed Intervals (Caché and Db4o).....	56
FIGURE 20	Preliminary Evaluation: Insertion at Fixed Intervals (Caché only)	56
FIGURE 21	Preliminary Evaluation: Find Different sized nodes.....	57
FIGURE 22	Preliminary Evaluation: Find a single node (Non-Cached Results)	58
FIGURE 23	Preliminary Evaluation: Find a single node (Cached Results)	58
FIGURE 24	Preliminary Evaluation: Find groups of nodes with in specified ranges	59
FIGURE 25	Preliminary Evaluation: Find groups of nodes (fewer configurations).....	59
FIGURE 26	Generation of RM Classes, Caché Classes, Proxy Classes and Conversion facilities	63
FIGURE 27	Contextual information that can be used to express paths as keys to object identifiers	64
FIGURE 28	Information to lookup any archetype node within LOCATABLE derived objects	66
FIGURE 29	Code to store a global structure using the initial single structure approach	66
FIGURE 30	Average time it takes to insert a single global node into the database as the time grows	67

FIGURE 31	Average index read/second for globals within different group sizes.....	67
FIGURE 32	Code to store global structures using indirection.....	68
FIGURE 33	Decomposing the search for archetype nodes in multiple steps using indirection	69
FIGURE 34	Global structure used to store path information of LOCATABLE objects in the prototype ...	70
FIGURE 35	Visual description of the data to be stored in the database for testing.....	72
FIGURE 36	Results from 'HD Tune' Benchmark for the Seagate hard drive	73
FIGURE 37	Database file size for 100 EHRs with 60 compositions.....	75
FIGURE 38	Storage space utilisation as the number of EHR objects grow in the database	76
FIGURE 39	Comparison of the average time to persist single types of compositions in the first test pass (<i>with standard error</i>).....	77
FIGURE 40	Comparison of the average time to persist a larger data set in to several <i>openEHR</i> implementations	77
FIGURE 41	Performance of Microsoft SQL Server queries on Composition Meta-Data.....	78
FIGURE 42	MS SQL (Fast Infosets): Avg. Time to retrieve a composition as the size of the database increases	79
FIGURE 43	Comparing the avg. time to retrieve a composition by UID in db4o and MS SQL.....	79
FIGURE 44	Performance of Microsoft SQL server for retrieving compositions based on archetypes	80
FIGURE 45	Comparative results for db4o and MS SQL on composition level queries.....	81
FIGURE 46	Performance of MS SQL (Fast Infoset): Content Queries in relation to the database size.....	81
FIGURE 47	Content at node id "at0004" within a specific archetype in the data set.....	82
FIGURE 48	Comparison of the average time to retrieve a single node from an archetype (<i>with standard error</i>).....	83
FIGURE 49	Code for setting up the logging facilities of a performance test	94
FIGURE 50	Simplified UML Diagram of Performance Monitoring Toolkit	95
FIGURE 51	.NET managed provider: Lists that have items which contain no objects	98
FIGURE 52	.NET Managed provider: One solution to the list problem, using a wrapper	99
FIGURE 53	Showing the Invalid Cast operation which the .NET Managed Provider threw.....	99
FIGURE 54	Db4o providing the ability to cast objects back to their original sub types	100

List of Tables

TABLE 1	Florescu and Kossmann (1999) mapping schemes for storing semi-structured XML.....	25
----------------	-----------------------------------------------------------------------------------	----

TABLE 2	Summary of comparisons in Van et al. between Hibernate/postgreSQL and db40.....	29
TABLE 3	Relevant system hardware and software specifications for testing environment	49
TABLE 4	Western Digital WD5000AAKB Hard Drive specifications for the preliminary evaluations	49
TABLE 5	Preliminary Test Configurations used to evaluate OODB's implementation featuresResults	55
TABLE 6	The set of compositions residing in each EHR in the data set.....	71

Acronyms

ADL	Archetype Definition Language
AQL	Archetype Query Language
AM	<i>openEHR</i> Archetype Model
ASN.1	Abstract Syntax Notation One
DB	Database (Used in reference to Objectivity/DB)
Db4o	Db4objects (Object Oriented Databases)
CEN	de Normalisation (European Committee for Standardization)
DTD	Document Type Definition
ECP	Enterprise Caché Protocol
EHR	Electronic Health Record
FI	Fast Infoset
GEHR	Good Electronic Health Record
GP	General Practitioner
HL7	Health Level Seven
MS SQL	Microsoft SQL (Server)
NEHTA	National E-Health Transition Authority
NHS	National Health Service
OACIS	Open Architecture Clinical Information System
OO	Object-Oriented
OODBMS	Object-Oriented Database Management System
OpenEHR	Open Electronic Health Records
QbE	Query By Example
RDBMS	Relational Database Management System
RM	<i>openEHR</i> Reference Model
SM	<i>openEHR</i> Service Model
SOA	Service-Oriented Architecture
SODA	Simple Object Database Access
SOAP	Simple Object Access Protocol (Not to be confused with Standardised Observation Analogue Procedure)
SQL	Structured Query Language
XML	eXtensible Markup Language
XML-QL	eXtensible Markup Language Query Language

Summary

Delivering optimal healthcare, particularly in areas such as integrated care, continue to be paralysed by a scattering of clinical information held across many incompatible systems throughout the health sector. The *openEHR* foundation develops open specifications in an attempt to mitigate the problem and finally achieve semantic interoperability, maintainability, extensibility and scalability in health information systems.

The *openEHR* architecture is based on a paradigm known as Two-Level Modelling. Knowledge and information is separated by forming a knowledge driven Archetype layer on top of a stable information layer. Clinicians create the domain concepts in archetypes which are used to validate information to be persisted at runtime. Archetypes impose a deep hierarchical structure on the information persisted in health systems.

Current known implementations of the persistence layer for *openEHR* use XML-enabled relational databases. Components of the EHR are stored and retrieved as XML files. Previous studies have shown that parsing and querying of XML files can impact on database performance. Mapping hierarchical data to relational tables is an option, but requires slow complex join operations. An object-oriented database is an alternative that may provide better performance and more transparent persistence.

This study compares and assesses the potential for the use of several Object-Oriented Databases in *openEHR* including Intersystem's Caché, Db4o and Objectivity/DB. The experience with using db4o and Intersystem's Caché including performance and implementation details are discussed. A tentative comparison with a current implementation of the *openEHR* persistence layer using Microsoft SQL Server 2005 is provided. This research's findings show that Object-Oriented database have the potential to provide excellent support for an *openEHR* persistence layer. However care needs to be taken in selecting the right OODBMS. The use of db4o or Caché, at least with the .NET managed provider cannot be advised for *openEHR* over the existing Microsoft SQL Server implementation with Fast Infosets.

Declaration

I declare that this thesis does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge it does not contain any materials previously published or written by another person except where due reference is made in the text.

Travis Muirhead

October 2009

1 Introduction

1.1 Background and Motivation

The scattering of information and incompatible systems throughout the health sector is limiting the capability of clinicians to provide optimal quality healthcare for patients (Conrick 2006). This inability to share health information seamlessly amongst healthcare providers or laboratories and separate departments within the same hospital reduces the capabilities or at least complicates decision support systems and other important aspects of integrated care (Austin 2004). Furthermore, many medical errors occur when information is not available at the required times. Classical examples include not knowing the history of adverse drug reactions and other complications that could even lead to death. (Bird, L, Goodchild & Tun 2003).

The problem described has been identified and understood for at least a decade (Hutchinson et al. 1996). Several standards organisations have been created and are working towards interoperable health records so information can be securely shared and understood between systems. Significant contributions have been made by organisations producing specifications such as *HL7* (HL7 2008), *openEHR* (openEHR 2008a) and *CEN* (CEN 2008). Although each organisation has similar goals for interoperability, their approach and scope differ

HL7 focuses on messaging to achieve interoperability between systems based on different information models and exchange of clinical documents. This type of messaging is important, but does not address other issues required to support the complexity and rapid creation or discovery of new knowledge in the health domain. The *openEHR* approach focuses on developing open standards for a health information system based on EHR requirements that addresses issues such as interoperability, maintenance, extensibility and scalability. *CEN*'s healthcare standards are focussed on communication and exchange of EHR extracts. *CEN* 13606 adopts the archetype driven approach developed for *openEHR* and in fact uses parts of the architecture defined in the *openEHR* standards (Schloeffel et al. 2006).

Support for the archetype driven approach in *openEHR* and *CEN* is quite widespread. For instance by 2007, *CEN* 13036 was being used in 48 different countries (Begoyan 2007). Interest in *CEN*

13036 lead to studies conducted by the UK's National Health Service (NHS) (Leslie 2007). The National E-Health Transition Authority (NEHTA) in Australia has analysed several standards for shared EHR and recommends the CEN13606 standard and points out the similarities to the *openEHR* approach (NEHTA 2007). Some companies and organisations have extensively used *openEHR* to build their Health Information Systems such as Ocean Informatics (Australia, UK), Zilics (Brazil), Cambio (Sweden), Ethidium (US) and Zorggemack (Netherlands).

A key point of interest in the *openEHR* specifications is the application of the approach known as two-level modelling. The two-level modelling approach incorporates two separate layers for information and knowledge. The information level is known as the Reference Model (RM) in *openEHR*. The RM is implemented in software and represents only the extremely stable, non-volatile components required to express anything in the EHR. The knowledge level is known as the Archetype Model (AM) in *openEHR*. The AM uses Archetypes which define concepts within the domain by only using the components provided at the information level in a structural arrangement required for that concept (Beale 2002).

This two-level modelling approach has significant advantages over a single-level modelling approach for the maintainability and interoperability of systems. Since domain concepts are expressed at the knowledge level, software and databases do not have to be modified to make changes, which are very important in a domain where new knowledge is constantly being discovered. Interoperability can be achieved by sharing a centralised archetype repository. Archetypes can also be defined to accommodate discrepancies in terminologies and language as they may be further constrained by templates for local use (Leslie & Heard 2006).

The *openEHR* foundation's technical activities work within a scientific framework (Beale et al. 2006). There has been significant research and published papers regarding the ontological basis for the architecture (Beale, Thomas & Heard, Sam 2007), the modelling approach (Beale 2002) and Archetype Definition and Query Languages (Ma et al. 2007). However there has been comparatively less studies focussing on the implementation aspects of the RM. The unique modelling approach incorporating archetypes imposes a hierarchical although somewhat unpredictable structure on the EHR. As a result the data being persisted at the RM level is structured very differently to conventional systems based on single-level approach. The

consequences of using specific database models on performance and efficiency are of interest to those implementing archetype driven software. This is especially the case in the health domain where large data sets from a patient's test results usually form a complex and deep hierarchical structure.

Due to the proprietary nature of several implementations of *openEHR*, information about current database models and packages in use is scarce but does exist. For instance trials such as the OACIS project and the GP Software Integration project focused on extracting data from non-GEHR (pre-cursor to *openEHR*) to conform to GEHR-compliant systems. The process generated XML-formatted files which are imported to a GEHR based repository (Bird, Linda, Goodchild & Heard 2002). Another approach used in a similar project LinkEHR-Ed also used XML mapping to make already deployed systems compatible with standardised EHR extracts. Essentially the LinkEHR-Ed data sources stay the same but place a semantic layer over the sources so data is provided to the user as an XML document (Maldonado et al. 2007). These approaches are also similar to the known approach used by Ocean Informatics, which is storing the fine-grained data as XML blobs in relational database tables with other higher level data. (*Ocean Informatics* 2008).

Using XML as an intermediate layer for the storage of data requires possibly 5 times more space (Austin 2004) and can also increase processing time. Attempting to store hierarchically structured and sparse data in a relational database without losing any semantics is costly for performance, coding time and integrity. Components from objects or tree-structures are split into many tables, and re-assembling the data with queries results in many join operations. This process is not an optimal way of processing data (especially as the database increases in size) and is a difficult case for programmers to manage complexity. (Objectivity, I 2005).

There are some alternatives to the previous approaches being used in *openEHR* projects that require either mapping or factorization of XML documents into relational databases. Object Databases or Post-Relational databases may provide better performance in *openEHR* systems without removing the semantics of the information model. Furthermore the object-relational impedance mismatch which exists in current implementations would be removed. This can lead to a reduction in development time and costs (Shusman 2002).

1.2 Research Questions

The main aim of this research is to investigate and compare the use of object-oriented databases to previous XML-enabled relational databases implemented in the persistence layer of an *openEHR* software project. The paper aims to answer the following questions:

1. Which database model is most semantically similar to the definitions provided in the *openEHR* Reference Model specification?
2. Can Object-Oriented databases provide the scalability, availability, security and concurrency needs of an *openEHR* based system?
3. Which database model provides the smallest amount of development effort?
4. What are the most suitable implementation technique using Object-Oriented databases for *openEHR* based systems?
5. How does an object-oriented database perform as compared to an XML-enabled relational database as the persistence engine in an *openEHR* software project?

Answering the questions above will assist developers of *openEHR* software systems in considering a database for their implementation.

2 openEHR Architecture

An understanding of the *openEHR* architecture is critical in order to evaluate the effectiveness of each database model and database that can be used for an *openEHR* project. This section presents the basis of the 2-level modelling approach and a high-level overview of the most relevant aspects of the architecture for finding the most appropriate database model.

2.1 Modelling and Design foundations

The *openEHR* modelling approach and architecture is based fundamentally on an ontological separation between information models and domain content models. The information model is stable and contains the semantics that remain static throughout the life of the Information System. The domain model is susceptible to change on the basis of new or altered knowledge within the domain. This separation helps enable future-proof information systems by accommodating change without having to change the information model, resulting in superior interoperability. It also results in a separation of responsibilities. For instance, in an *openEHR* based information system, IT professionals build and maintain the information model, whilst the clinicians create and manage the domain knowledge. A paradigm based on this ontological separation has become known as Two-Level Modelling (see FIGURE 1). (Beale, T & Heard, S 2007c)

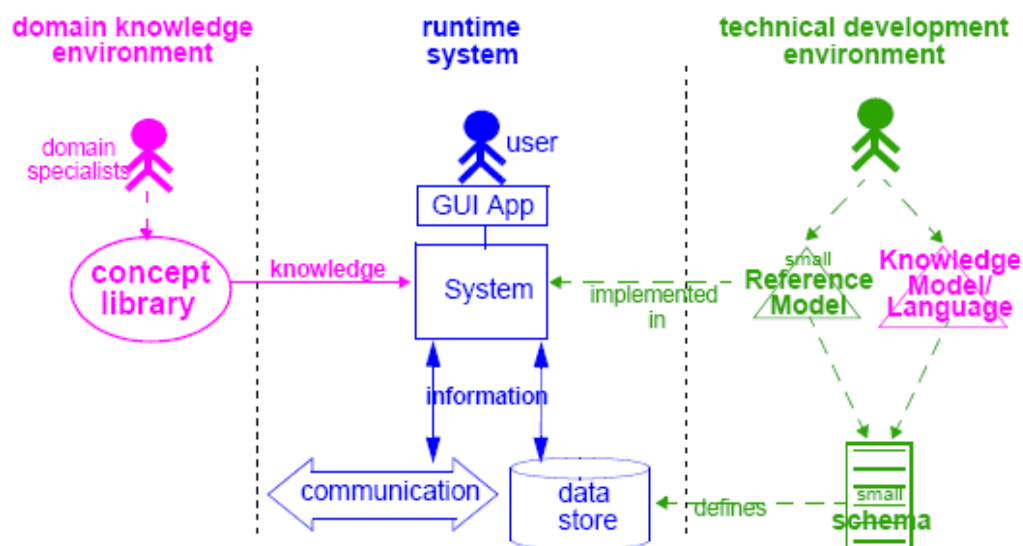


FIGURE 1 A Two-Level Modelling Paradigm

(Beale, T & Heard, S 2007c) © Copyright *openEHR* Foundation 2001-2004. All rights reserved. www.openEHR.org

The result of this type of ontological separation results in a vastly different approach to the more traditional and popularised approaches described in object-oriented methodologies (Larman 2005) or relational database texts (Connolly & Begg 2005) incorporating data or information modelling techniques. The approach described in the aforementioned texts result in a single-level modelling paradigm (see FIGURE 2) where there is no ontological separation between domain content models and information models. Instead, domain concepts are incorporated into the information model which is implemented directly into software and databases. In such systems, maintenance becomes frequent and also problematic because the introduction of new domain concepts requires structural changes that make it more difficult to achieve interoperability. The single-level approach may work well in systems with low complexity and minimal changes in domain knowledge, but the situation is different in the health domain. (Beale 2002) It is worth noting that many information systems do not rely on this traditional approach. Although there is an ongoing trend in computing to further abstraction and in recent decades a number of newer model-driven engineering approaches have emerged which may be similar.

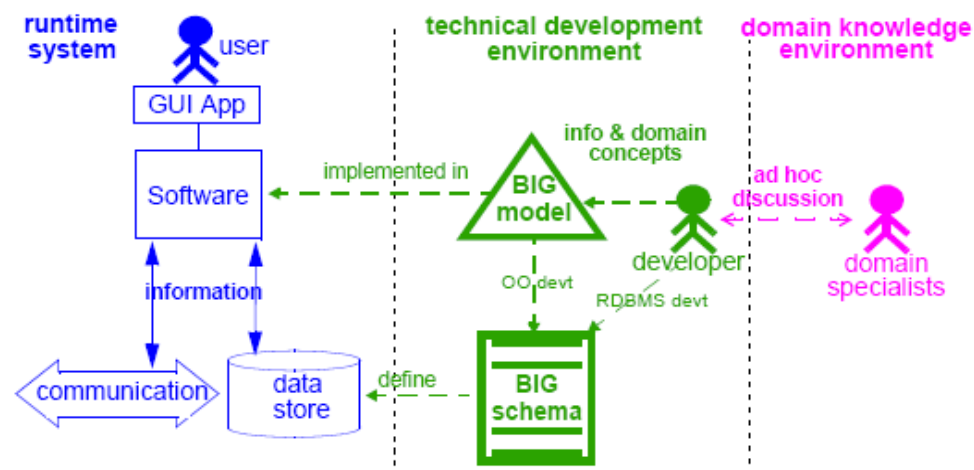


FIGURE 2 A Single-Level Modelling Paradigm
(Beale 2002) © Copyright Thomas Beale 2002

2.2 Package Overview

There are three main outer packages in the *openEHR* architecture which can be seen in FIGURE 3. The reference model (RM) is at the information level and implemented in software and databases. The archetype model (AM) is at the knowledge level and contains semantics required to support domain knowledge in the form of archetypes and templates. The Service Model (SM) provides services that interface with the EHR and external archetype repositories and terminologies. (Beale, T & Heard, S 2007c)

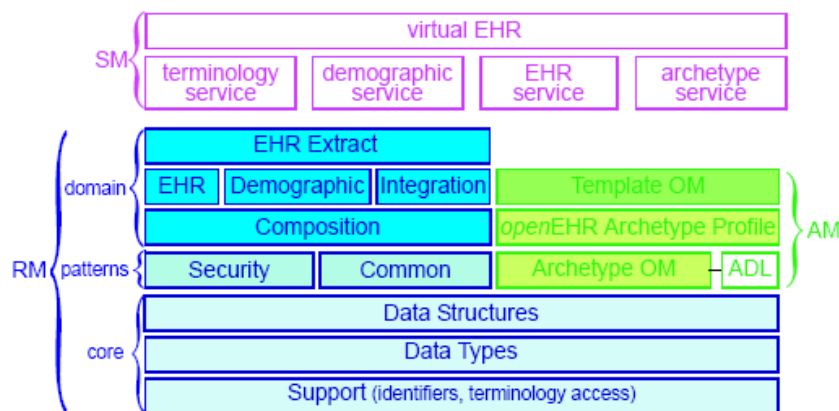


FIGURE 3 *openEHR* package structure

(Beale, T & Heard, S 2007c) © Copyright *openEHR* Foundation 2001-2004. All rights reserved. www.openEHR.org

2.3 Archetypes and Templates

An archetype is a structured formal definition of a single domain concept. In terms of the healthcare domain, concepts such as “blood pressure”, “cardiovascular examination” or “prescription” can be defined as archetypes (*openEHR* 2008b). There exists a set of principles which define an archetype (Beale, T & Heard, S 2007b). An important collection of principles in understanding the role of archetypes in information systems such as those based on *openEHR*, is that archetypes define constraints on the information expressed in the information model. This principle can be visualised in FIGURE 4, showing how information is validated at run time against archetypes before it is persisted in storage. It also shows the separation of archetypes from the information model. (Beale, T & Heard, S 2007b)

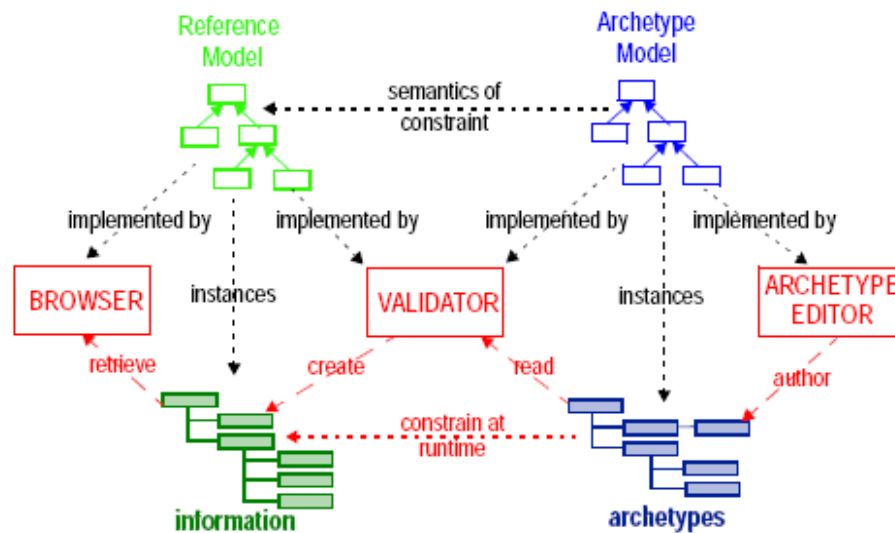


FIGURE 4 Archetype Software Meta-Architecture.
(Beale 2002) © © Copyright Thomas Beale 2002

Other principles define how an archetype is actually composed. For instance, archetypes inherently form a hierarchical tree structure due to the object model. Furthermore, archetypes can be composed of other archetypes or specialised, inheriting its basic structure from parent archetypes (Beale, T & Heard, S 2007b). An example of the hierarchical nature of archetypes can be seen in FIGURE 5 which contains a partial node map of an archetype for laboratory results. The node map was generated by an archetype editor tool which converted a representation of the archetype as Archetype Definition Language (ADL) (Beale, T & Heard, S 2007a) to the node map.

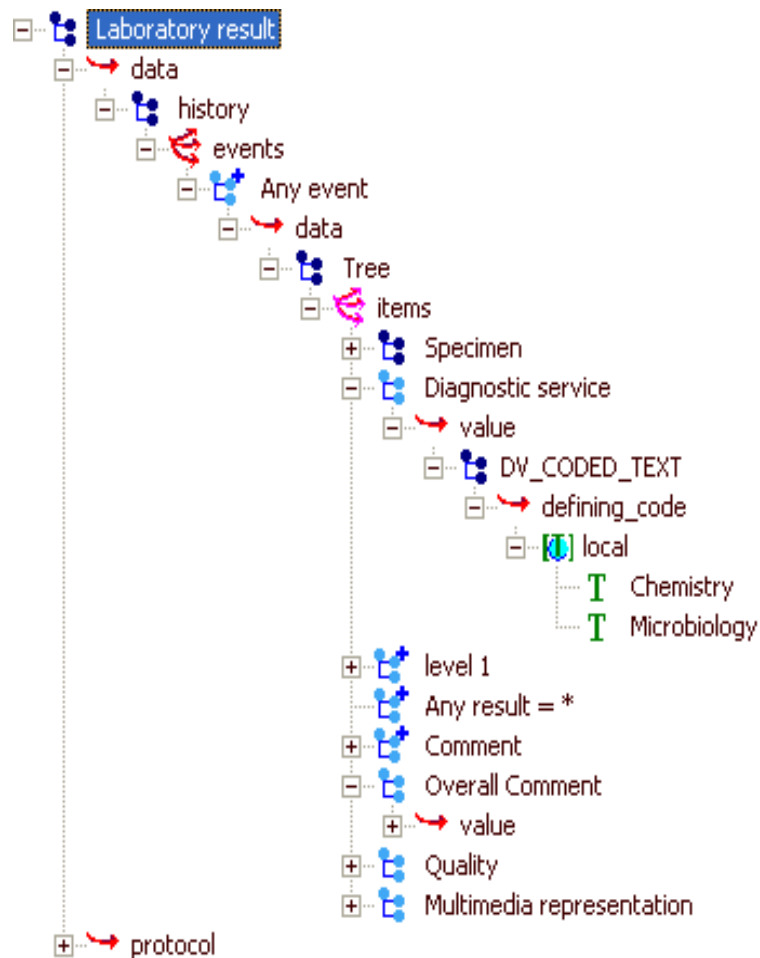


FIGURE 5 Partial node map of an archetype for laboratory results

Archetypes are designed for wide re-use and the ability to share information. In local healthcare settings, the archetypes may need to be further constrained to the preferences of the clinicians at that point-of-care and several archetypes need to be combined to facilitate a complete composition in the EHR. An *openEHR* Template can be defined locally to conform to these properties. Templates usually correspond directly to the forms clinicians will be using in the system. (Beale, T & Heard, S 2007d)

2.4 Structure of the EHR and Compositions

Each representation of an Electronic Health Record (EHR) in *openEHR* contains an identification value which is globally unique. The unique identifier may be used across several *openEHR* systems or distributed systems, enabling reconstruction of a more complete health record from several EHR repositories. The top-level EHR structure is also composed of information such as access control, status, compositions, contributions and a directory which can be seen in FIGURE 6. Many of these structures or containers within the EHR are subject to version or change control in accordance with the requirements for the EHR. (Beale et al. 2007)

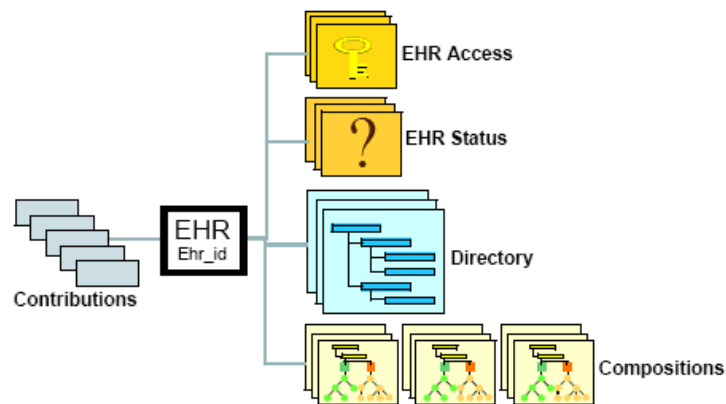


FIGURE 6 High-Level Structure of the *openEHR* EHR

(Beale et al. 2007) © Copyright *openEHR* Foundation 2001-2004. All rights reserved. [www. openEHR.org](http://www.openEHR.org)

The composition object is of most interest in understanding the structure of data stored in an *openEHR* system. A new composition corresponds to each clinical statement to be recorded in the EHR. Compositions are versioned such that changes to the EHR result in new compositions managed in Versioned Composition objects, rather than simply modifying the contents of the original composition (Beale, T & Heard, S 2007c). The overall structure of a composition is shown in FIGURE 7.

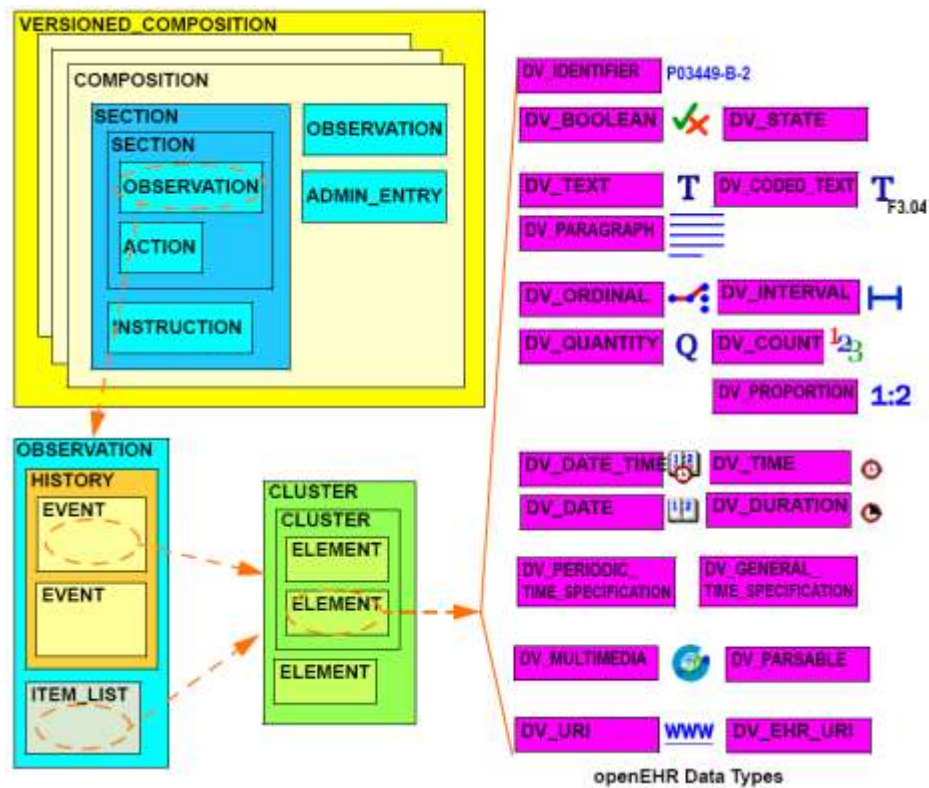


FIGURE 7 Elements of an *openEHR* Composition

(Beale, T & Heard, S 2007c) © Copyright *openEHR* Foundation 2001-2004. All rights reserved. www.openEHR.org

The composition object can contain either no items or a combination of one or more "Section" and "Entry" objects which both inherit their attributes from a class called CONTENT_ITEM. Sections group entries into a logical structure or can also contain more Section objects, creating deeper tree structures. The entries correspond to the Clinical Investigator Record Ontology which describes a model of the types of data clinicians capture in the evaluation of health status (Beale, Thomas & Heard, Sam 2007). There are four subtypes of care entries that exist in *openEHR*: observation, evaluation, instruction, action (See FIGURE 8).

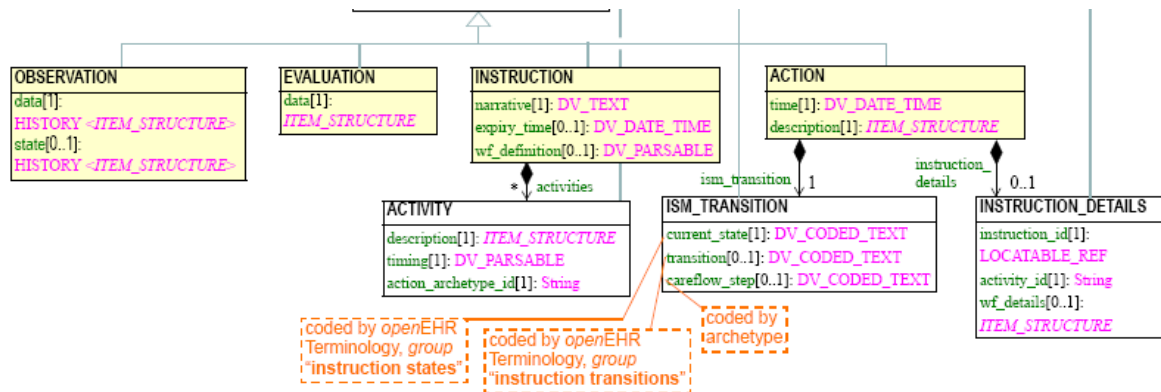


FIGURE 8 Partial view of the entry package, showing the subclasses of CARE_ENTRY
(Beale et al. 2007) © Copyright *openEHR* Foundation 2001-2004. All rights reserved. www.openEHR.org

The care entries extend the tree structure further, each containing lists that can potentially contain more nested lists. This can be seen in FIGURE 7 where an observation entry contains a list of events which can reference a cluster which can include another reference to another cluster. This type of structure is similar to the composite design pattern (Gamma et al. 1994) which is prevalent in folder structures and scene graph or transform groups in graphics applications.

2.5 Paths, Locators and Querying

The common information model provides a class called *PATHABLE* which most classes in the *openEHR* reference model inherit attributes and functionality from. This enables a path mechanism that can be used for locating data in the system. A query language was developed called EHR Query Language (EQL) as a standardised approach for querying data in *openEHR* and other archetype based systems (Ma et al. 2007). EQL has been extended and renamed as Archetype Query Languages (AQL) (Beale 2008). A component is needed to process AQL queries and convert them to a form which can be used by databases. Such a component exists for the Ocean Informatics implementation but new mappings may be required for a different database.

3 Database Models

This section reviews three database models and compares their suitability for managing complex data such as healthcare data in *openEHR*.

3.1 Relational Databases

The relational database was introduced as an alternative to early Hierarchical or Network database models. These early models were fast but suffered from a lack of structural independence between the physical and logical view of the data. Any changes to the internal representation would affect both users and applications that access the database (Conrick 2006). A relational model based on set theory was proposed with an emphasis on providing structural independence (Codd 1970).

A relation is an unordered set of n -tuples. This statement can be visualised graphically as a two-dimensional table where each row corresponds to a tuple and each column is the elements of the tuple within the same domain. Each tuple in the relation should be unique such that a domain or combination of domains forms a primary key. The same combination of domains that form the primary key can appear in another relation as a foreign key such that the primary key cross-references the foreign key forming a relationship. Operations can be applied to one or more relation. Projection operations are used to select specific domains from a relation and a join operation can be used to combine two or more relations with mutual domains, such as the primary to foreign key relationship described. (Codd 1970).

Since the proposal of the relational model, many database management systems have incorporated the same concepts and principles including well known proprietary databases including Oracle, IBM DB2 and Microsoft SQL. The latter is used by Ocean Informatics in their current implementation of the *openEHR* Reference Model (Ocean Informatics 2008).

After the introduction of the relational model, it became clear that it presented challenges in representing the semantics of information models (Codd 1979). Despite new extensions to the relational model, the tabular format of data presents issues with representing complex objects. This is especially true for objects comprised in a hierarchical structure.

The simplest method of modelling complex objects in a relational database is to store instances of each smaller object composing the complex object in separate tables. In order to enforce the semantics a stored query can be used to re-construct more complex objects from their composed parts. However, performance becomes an issue because many join operations are required each time a user wants to re-construct the complex objects. Join operations are an expensive operation in comparison to object references which can be seen in FIGURE 9. Significant research has been undertaken into optimising join operations to provide better performance (Priti & Margaret 1992), however the space for improvement is limited.

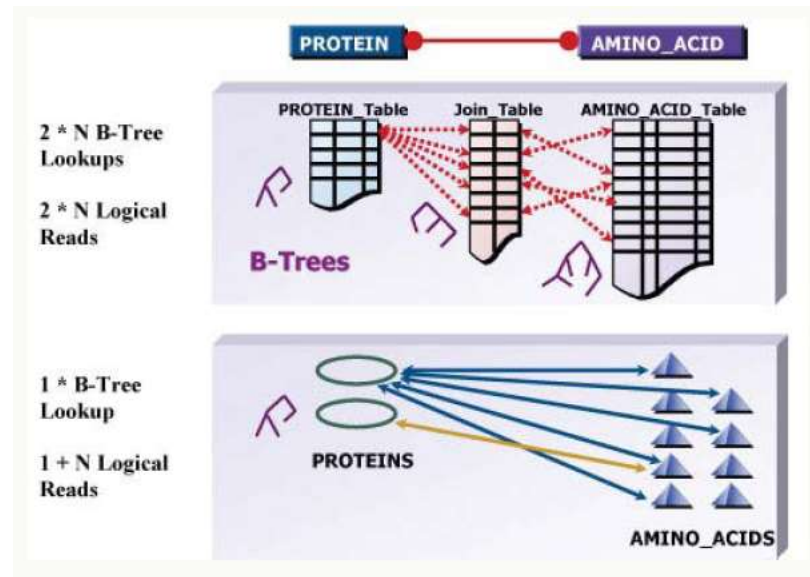


FIGURE 9 Comparison of join operations in an RDBMS to references in an OODBMS (Objectivity, I 2007)

The overhead created by join operations may be able to be removed if the complex object is flattened into a single table. However this introduces other issues such as data redundancy, integrity and disk space requirements. Instances of smaller objects which could have been re-used now must be repeated. Multiplicity also results in many NULL or empty attributes as total number of columns is needed to satisfy the multiplicity constraints, but some objects may store fewer values (Objectivity, I 2007).

Hierarchies can also be modelled by representing the structure with adjacency lists, nested sets and other data structures (Celko 2004). However the semantics become very difficult to manage and the performance is not optimal.

3.2 XML enabled Relational Databases

Several relational databases now include support for storing XML. The motivation for storing XML on relational databases varies. XML is being widely used for data-exchange in distributed systems and there has been interest in storing XML in existing relational databases for this purpose (Bauer, Ramsak & Bayer 2003). Others have acknowledged the issues of storing semi-structured data in relational databases and present the storage or mapping of XML files in relational databases to enable storage of semantically complete tree structures (Khan & Rao 2001). Typically health data is semi-structured which has led to the use of XML files for organising and storing data in health information systems. This section investigates some of the approaches used for mapping XML to a relational database and the implications on the database performance and capabilities.

Early methods used to support XML in relational databases relied on mapping XML data onto relational tables. This process is known as XML Publishing for re-constructing XML documents from a database and XML Shredding for decomposing XML documents on relational tables (Rys 2005). Many different mapping schemes have been proposed for this purpose with varying results in performance and usability.

Florescu and Kossmann (1999) compare five mapping schemes for storing semi-structured XML documents within a relational database. Semi-structured XML data does not conform to a particular schema as it may be changing. Florescu and Kossmann use a directed labelled graph for representing the XML data. Nodes provide the object ID; edges represent attributes or elements and the leaves contain the data values. A summary of the mapping approaches used are shown in TABLE 1. The key of each relational table is denoted in **bold** and the index is denoted in *italics*. The relational tables are in the form: table (column₁, column₂, column_n). The flag columns denote whether the attribute references a value or another object.

Approach	Structure	Comments
Edge	Edge (<i>source</i> , <i>ordinal</i> , <i>name</i> , <i>flag</i> , <i>target</i>)	Uses only one table to store all edges. There is a combined index on name and target.
Attribute	A _{name} (<i>source</i> , <i>ordinal</i> , <i>flag</i> , <i>target</i>)	A table for each attribute is created
Universal	Universal(<i>source</i> , <i>ordinal</i> _{n1} , <i>flag</i> _{n1} , <i>target</i> _{n1} , <i>ordinal</i> _{n2} , <i>flag</i> _{n2} , <i>target</i> _{n2} , ..., <i>ordinal</i> _{nk} , <i>flag</i> _{nk} , <i>target</i> _{nk})	Stores all attributes in one table. 3 columns used to denote a single attribute (<i>source</i> , <i>ordinal</i> _{nk} , <i>flag</i> _{nk} , <i>target</i> _{nk}). This approach is not normalised and thus results in redundant data.
Normalised Universal Approach	UnivNorm(<i>source</i> , <i>ordinal</i> _{n1} , <i>flag</i> _{n1} , <i>target</i> _{n1} , <i>ordinal</i> _{n2} , <i>flag</i> _{n2} , <i>target</i> _{n2} , ..., <i>ordinal</i> _{nk} , <i>flag</i> _{nk} , <i>target</i> _{nk}) Overflow _{n1} (<i>source</i> , <i>ordinal</i> , <i>flag</i> , <i>target</i>), ... Overflow _{nk} (<i>source</i> , <i>ordinal</i> , <i>flag</i> , <i>target</i>)	Similar to the approach above but overflow tables are used to normalise the approach. An extra value for flag is used "m" to denote if attribute has multiple values.
Separate Value Tables	V _{type} (<i>vid</i> , <i>value</i>)	Is to be used with previous techniques, extending them to allow use of different data types.
In-lining	Used with first 3 techniques by replacing the flag with the values. This approach is not normalised.	

TABLE 1 Florescu and Kossmann (1999) mapping schemes for storing semi-structured XML

Florescu and Kossmann (1999) queried the database by translating XML-QL into SQL queries. Typical comparisons were made on the time to reconstruct objects from the XML, selection on values, optional predicates, predicates on attribute names and regular path expressions. Basic operations such as updates, insertion of objects and deletion of objects are also timed and compared. An XML document with 100,000 objects, maximum of 4 attributes per reference and 9 attributes per object is used. Unfortunately the document only contains 2 data types which are not substantial. The size of the document is quite large at 80mb.

Florescu and Kossmann (1999) made several observations that provide reason to avoid storing XML in relational databases. Although query processing was reasonably quick, reconstruction of the XML document was extremely slow. For every mapping scheme it took at least 30 minutes to reconstruct the object. Also some important database features including concurrency was not

addressed by Florescu and Kossmann . The authors concluded that the in-line approach performed best; however the time taken to reconstruct objects from in-lined tables is still poor.

A study by Tian et al. (2002) extended the work by Florescu and Kossmann (1999) and Shanmugasundaram et al. (1999) The study compared 3 approaches: Relational DTD, object approach and the edge/ attribute approach. The *relational DTD* approach maps DTD elements to a set of tables in which each table contains an ID and parentID column (Shanmugasundaram et al. 1999). The *object approach* does not use a relational database; instead it uses an object manager holding XML elements as light weight objects inside of a file object representing the complete XML document. Performance evaluation showed that the object approach was 40 times faster than the attribute approach where as the DTD approach was only marginally faster than the edge approach. However the DTD approach performed better at processing queries.

Bauer et al. (2003) presents a Multidimensional Mapping and Indexing of XML documents. The concept is based on a model built on three main ideas from an XML document; Paths, Values and Document Identifiers. The implementation of their approach is based on Multidimensional Hierarchical Clustering (MHC) described in Markl et al. (Markl, Ramsak & Bayer 1999). The schema implementing this technique includes two tables: xmltriple and typedim. The table xmltriple includes 3 columns: did, val and surr. The table typedim has 2 columns: surr and path. Values are stored in xmltriple and correspond to paths in the typedim tables using a concept known as a binary encoding of a composite surrogate. The results of this study show that this approach with a combination of B-tree indexes provides vastly superior results in selection and projection to the edge mapping approach defined by Florescu and Kossmann (1999). Unfortunately, performance of reconstruction of XML documents or objects is not discussed.

The method for storing XML in Microsoft SQL Server 2005 is discussed in Rhys (2005). Whilst the early approach of XML publishing and shredding is still available (Microsoft 2007), Rhys discusses the addition of native XML support to the database. Although support to persist XML in BLOBs was available, this extension includes semantic validation and allows processing and querying of the XML data. As XML documents are persisted they are parsed and validated to a XML schema from a collection of schemas stored in the database. The overall approach can be seen in FIGURE 10. Implementation of the subset of XQuery, focuses only on one data type within a single query.

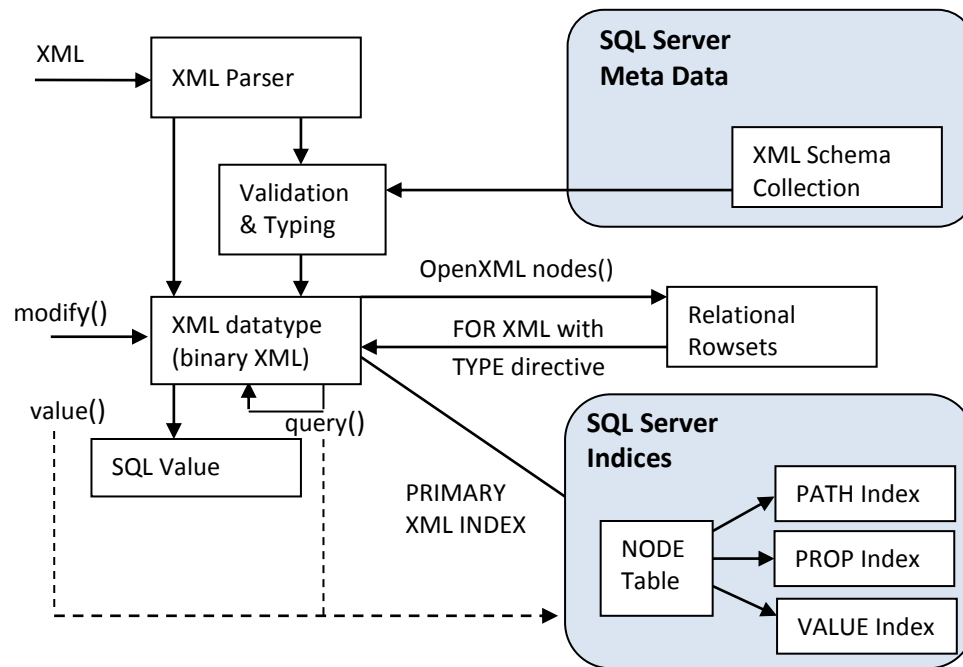


FIGURE 10 SQL Server 2005 XML architecture overview
(Rys 2005)

Further limitations of approaches used in Microsoft SQL Server 2005 are described by Nicola and John (2003). Although the paper was presented prior the deployment of Microsoft SQL Server 2005, it focuses on the negative impacts of XML parsing on database performance. The only time parsing is avoided is when an XML file is persisted simply as a character array or CLOB without any of the additional database management features on the actual structure of the XML document. Regardless of the technique used in current implementations in *openEHR*; parsing, validation, querying, concurrency and other important features need to be implemented even if it is in the front-end. The findings of Nicola and John (2003) show that parsing larger files with validation decreases the relative overhead. However for a 1MB XML document, XML schema validation adds 90.94% overhead while the DTD validation adds 20.52% overhead. The work was not performed on binary XML and the study identifies the need for further research in this area.

Fast Infosets provide another alternative to the native XML support provided in Microsoft SQL server (Sandoz, Triglia & Pericas-Geertsen 2004) which is based on a binary encoding that may improve the performance of XML processing. Fast Infosets are used in the *openEHR* persistence layer that Ocean Informatics have implemented with Microsoft SQL Server 2005 and it was found

that this method performed better than using the native XML facilities provided by the database itself. This can be illustrated by the results obtained by a set of Fast Infoset Performance Benchmarks performed by Noemax (Noemax 2009). The results show over 140% increase in performance using the Fast Infoset reader/writer for both text and .NET binary and almost 600% performance increase over text using the Fast Infoset dictionary reader/writer with SAX1. Furthermore, size comparisons performed by Noemax show that the Fast Infosets dramatically reduce database size. Data from a worldwide protein bank stored in a Fast Infoset was typically around 7.7 times smaller than text and 3 times smaller than the .NET binaries.

3.3 Object-Oriented Databases

Atkinson et al. (1989) put forward a possible standard of an object-oriented database system which has often been used as the basis for discussing object oriented databases. The paper describes what should be mandatory for an object database to include as well as other optional features. The features which are described of as being mandatory are a mixture of traditional object-oriented programming paradigms with the transactional features provided by most database management systems.

Since object-oriented languages are in wide use now, databases providing object access have the potential to provide an explicit one-to-one direct mapping with programming data structures. Object oriented databases naturally provide a greater level of semantic expressiveness and robustness due to the object compositional aspects. Atkinson et al. (1989) still mention the requirement of an Ad Hoc Query Facility, providing developers with the ability to decide on the granularity of access.

The most obvious advantage of Object-Oriented Database Management Systems (OODBMS) is the transparency between the database and object oriented programming languages such as Java, C# and Smalltalk. Relational databases do not have this transparency due to their declarative and set-oriented nature (Chaudhri 1993). This makes it difficult to store object instances to a relational database and usually requires mappings that reduce overall performance.

Evidence supporting the Object-Relational Mapping (ORM) overhead is published in Van et al. (2006). The study compares the performance of hibernate (an ORM) on postgresQL to db40 (an

OODBMS). A complete benchmarking database (OO7) was created which includes complex objects and deep hierarchical data from an engineering design library. The main operations tested were creation, traversal, modification and queries. It was found that db4o was significantly faster in all operations. The operations tested are summarised in TABLE 2.

OPERATION	DB4O	HIBERNATE/POSTGRESQL
Creation	76 seconds	198 seconds
Traversal T3C (40,000 objects)	25 milliseconds	75 milliseconds
Query Q7 (finds all 10000 atomic parts)	20 milliseconds	275 milliseconds
Insertion (hot run)	12.0 seconds	23.4 seconds
Deletion (hot run)	13.3 seconds	18.7 seconds

TABLE 2 Summary of comparisons in Van et al. between Hibernate/postgreSQL and db4O (Zyl, Kourie & Boake 2006)

There are several areas where an object database should provide immediate gains in performance over pure relational databases and XML-enabled databases for storing EHR data. An object oriented database typically uses references inside complex objects to point to the other objects it is composed of. Instead of performing a search and compare routine on separate database tables, the location of the classes inner objects and data members are already known and can be fetched directly (Objectivity, I 2007). Also, due to the transparency between the object oriented front-end and the database, the additional mapping, serialisation and validation is not needed as compared to the XML-enabled relational database approach described earlier.

The level of transparency between the application code and the Object Oriented database depends on the OODBMS product. For instance, Intersystems (Intersystems 2009e) provides a tool for Caché to automatically create proxy classes in some programming languages including C# and Java. However other databases such as db4o provide even greater transparency. In db4o, there is no need for proxy classes, objects can be persisted by a simple method call Store(Object o).

4 OODB Products and Technologies

This section provides an overview of the features and technological characteristics of some OODBMS database products currently available on the market. There are a large number of products on the market and a detailed analysis of each product is out of the scope of this study. For each particular database model, the particular set of features, licensing options and scale for each product discussed are considerably different. Furthermore the level of transparency between the programming language and each database discussed varies even for each ODBMS.

From a purely semantic view point it is quite clear that the relational model is not the best solution for *openEHR* data. Review of previous literature shows there is also a compelling reason not to store XML in a relational database for reasons of performance, development time and maintainability. However there is a range of other issues to consider. The *openEHR* community raised a question whether an OODB can, in practise, provide acceptable performance. Each database is aimed at a different niche but previous case studies have shown that certain databases using either model can perform well even on complex object structures and has the ability to scale to the needs of a very large *openEHR* based system.

This section compares three Object-Oriented databases: db4objects, Intersystem's Caché and Objectivity. These databases have been selected for their interoperability with C# .NET and wide range of features and licensing options.

Other potential databases which could be used for *openEHR* include XML-Relational databases (as mentioned in SECTION 3) and newer technologies such as native XML storage systems. However the XML-Relational databases (including Microsoft SQL Server and Oracle) have the drawback of parsing as mentioned in SECTION 3 as well as a lack of control of the locking granularity and constraints. Native XML storage systems are relatively new although some native XML databases such as XTC (Kaiserslautern 2009) has addressed many XML storage issues and there is a large group of people working to resolve the remaining problems. This type of database is very promising and may be useful for certain applications such as document-centric applications.

These native XML database may assist with the parsing problems in a similar way as Object-Oriented databases, but are not the focus of this paper and can be left for investigation in other studies.

4.1 dB4objects

dB4objects (dB4o) is a Native Java/.NET open source Object-Oriented database (Versant 2009). The features provided by db4o are extremely ambitious. Persisting objects to db4o is very simple as there is no mapping between the programming language and the data model. Due to its small footprint it is particularly useful in embedded software. However it lacks support for large scale applications.

4.1.1 Storing Objects using C#

The db4o API is packaged into a single .DLL/.JAR file which can be referenced by a project created in an IDE or added to the class path. There is no need to generate a separate schema for db4o as it uses reflection to create the data structures for storage from the class definitions of the objects you persist. FIGURE 11 shows the simplicity of storing an EHR instance in a db4o database in the C#. These examples have been simplified and do not show how each of the parameters of the EHR extract are instantiated.

```
...
IObjectContainer db = db4o.Factory.OpenFile(db.yap);
try {
    EHR ehr = new EHR(system_id, ehr_id, time_created,
        ehr_access, ehr_status, directory, compositions, contributions);
    db.store(ehr);
} finally {
    db.close();
}
```

FIGURE 11 Persisting objects in db4o with C#

The first commit will typically be slower as it needs to configure the database. Transactions are committed implicitly by default, but the feature to explicitly commit a transaction is available in situations such as the storage of large data sets. There is essentially no difference in persisting objects in a client/server setup. If the *openEHR* reference model has already been implemented in C# or java, persisting objects with db4objects is very simple from a development perspective.

4.1.2 Retrieving Objects using C#

There are several ways objects can be retrieved in db4o such as Query by Example (QbE) a derivative of the technique outlined by Zloof (1975), Simple Object Data Access (SODA) queries and Native Queries (Cook & Rosenberger 2006). None of these use a string based approach like SQL. QbE is limited in what it can do and SODA has been superseded by Native Queries as the preferred query mechanism for db4o. There are some situations where SODA might be preferred due to optimisation issues which are actively being addressed by the db4o development community.

After some experimentation, it was found that Native Queries do not currently provide the necessary performance for *openEHR*. A simple test storing a linked list with 1 million nodes (each labelled with a unique number) was committed to the database. Searching for a particular node was faster with SODA queries but less elegant. one of the problem causes might be problematic settings of the native query optimiser. FIGURE 12 shows a typical AQL query (the standard query language for *openEHR* and the equivalent query using SODA in FIGURE 13. However the query in FIGURE 13 returns the whole composition and not just the range of values requested which may result in unnecessary disk access.

```
SELECT
o/data[at0001]/events[at0006]/data[at0003]/items[at0004]/value AS Systolic,
o/data[at0001]/events[at0006]/data[at0003]/items[at0005]/value AS Diastolic
FROM EHR [ehr_id=$ehrUid]
      CONTAINS COMPOSITION c[openEHR-EHR-COMPOSITION.encounter.v1]
      CONTAINS OBSERVATION o[openEHR-EHR-OBSERVATION.blood_pressure.v1]
WHERE
o/data[at0001]/events[at0006]/data[at0003]/items[at0004]/value/value >= 140
OR
o/data[at0001]/events[at0006]/data[at0003]/items[at0005]/value/value >= 90
```

FIGURE 12 A typical AQL query
(Ma et al. 2007) © Copyright Ocean Informatics

```

...
IQuery query = db.Query();
query.Constrain(typeof(COMPOSITION));

IQuery compQueryCheck = query.Discard("archetypeNodeId")
    .Constrain("openEHR-EHR-COMPOSITION.encounter.v1");

IQuery obsQuery = query.Discard("content");
obsQuery.Constrain(typeof(OBSERVATION));

IQuery obsQueryCheck = obsQuery.Discard("archetypeNodeId")
    .Constrain("openEHR-EHR-OBSERVATION.blood_pressure.v1");

IQuery systolic = obsQueryCheck.Discard("data");
systolic.Discard("events").Discard("data").Constrain(typeof(ITEM_TREE));
systolic.Discard("items").Discard("archetypeNodeId").Constrain("at0004");
systolic.Discard("value").Constrain(typeof(DV_QUANTITY));
systolic.Discard("value").Discard("units").Constrain(140).Greater();

IQuery dystolic = obsQueryCheck.Discard("data");
dystolic.Discard("events").Discard("data").Constrain(typeof(ITEM_TREE));
dystolic.Discard("items").Discard("archetypeNodeId").Constrain("at0005");
dystolic.Discard("value").Constrain(typeof(DV_QUANTITY));
dystolic.Discard("value").Discard("units").Constrain(140).Greater();

IObjectSet result = query.Execute();

```

FIGURE 13 Retrieving objects from db4o with C# and SODA queries

Retrieving a structured object may load the complete object tree hierarchy into memory, including nodes which are not required. By default db4o does not load objects that are referenced in the object being retrieved. However, the API provides methods which control the depth of the referenced objects to retrieve. This is known in db4o as setting the “Activation Depth”. This feature is absolutely necessary for Object-Oriented database systems to provide good performance. SECTION 5 further discusses the implementation techniques in db4o and explains why it is only feasible to include path information in the objects being searched for or a separate structure for queries.

4.1.3 Storage Capacity

According to db4o (2009), each database file can hold only 254 GB of data which is extremely small compared to some other solutions. For instance Objectivity/DB has been deployed in High Energy Physics systems managing almost a petabyte of data such as at the Stanford Linear

Accelerator Centre for BaBar (Becla & Wang 2005). Although a vast amount of code was produced to customise Objectivity/DB to allow BaBar to scale to that magnitude.

The block size of the database set in the configuration determines the maximum capacity of the database. Versant (2009) recommends that their other databases be considered if the database is to be designed to store over 10GB. Paterson (2006) explains that the optimal block size is 8 bytes which corresponds to a maximum database file of 16GB. This is quite small for certain healthcare systems considering some of the contribution objects measured for *openEHR* have been up to 200 MB each for particular laboratory results. It is possible to reduce the size of the stored database considerably by using db4o BLOBs for large objects which is stored separately from the database file. In contrast with relational databases, db4o BLOBs can still be accessed through the object tree.

4.1.4 Maintenance

Very few tools are provided for maintenance and administration activities other than a defragmentation tool. This partially reflects the databases' ability to be distributed in embedded environments where administration is very difficult. In many ways the database is capable of running without administration; however defragmentation is required for best performance. Unfortunately db4o cannot run defragmentation when the database is live. As a result, it is not suitable for enterprise healthcare systems and other systems that require reliability and no down time.

Contrasting the limitations mentioned above, db4o is a very mature tool for minimising development time during software maintenance. Due to the tight integration with the programming language, updating the classes and source code automatically update the database schema. Data from previous versions will still be compatible with the new changes.

4.1.5 Concurrency

The database does not provide many features which automatically allow the product to scale and handle many users. In, fact Versant recommends using their commercial database for requirements of handling more than 20 connections at once instead of db4o. It is also up to the programmers to manage safe concurrency as db4o only provides overly optimistic locking which does not include

collision detection. Furthermore, the programmers also have to manage connection pools. This makes the product flexible but would increase development time significantly.

4.1.6 Security

Due to the nature of healthcare data, *openEHR* has identified rather strict security requirements which are discussed in Beale T & Heard (2007c). Many of the basic security features are addressed in the *openEHR* specification and would be included in the application logic rather than the database such as access control and logging. Some of the specific details of security are yet to be addressed in the *openEHR* specification; however there is a foundation of work recommending digital signatures and encryption. Of course each type of system will have very specific details, for instance, securing wireless connections and indeed preventing physical access.

It may be necessary to implement some of the features specified for *openEHR* in the database to ensure security, particularly encryption. Db4o provides the eXtended Tiny Encryption Algorithm (XTEA). Recent cryptanalysis on this algorithm has shown a theoretical 80% probability of breaking the encryption with a Related Key rectangle attack on 36 rounds in $2^{104.33}$ time complexity (Lu 2009). This provides strong encryption and may be suitable for *openEHR* databases.

4.1.7 Distribution

At present, db4o does not provide the necessary facilities to create a federated and highly distributed database solution that may be needed in hospitals. Instead, db4o possess the characteristics that allow it to be used in embedded applications such as the small foot print. The .NET 2.0 deployment of the main db4o package .DLL at this point in time is less than 800 kb. Further support for distribution may be included in future versions (Paterson 2006).

In order to make db4o scalable and handle many of the issues required by a distributed system needs to be implemented by programmers. This is a large undertaking but may be useful if an extremely customised solution is needed. Whilst there are specifications regarding security, concurrency and time in *openEHR*, recommendations for best practises or specifications for deployment as a distributed system does not exist. As such, it may be better to consider other solutions for anything other than mobile healthcare or other small deployment used in conjunction

with a SOA and the “dRS Replication System” provided by db4o for synchronisation with other data sources.

4.1.8 Fault Tolerance and Availability

Programmers are able to use the dRS Replication System to replicate a db4o data store. This mechanism can switch to a failover database when a fault occurs. However, additional work is required to configure the dRS Replication System to support the requirements of a large health system.

4.1.9 Support

The database is an open source initiative and all of the source code is available to developers. A network of over 60'000 developers are involved in developing the software. Versant is currently the major sponsor of the initiative. A complete tutorial and API reference guide is included in the documentation. Some hints and optimisation techniques are available in the reference documentation, but it is considerably smaller than some other database vendors. However this may reflect the size and nature of the database.

4.1.10 Opportunities

Out of the box db4o is not suitable for many of the requirements that are specified by large healthcare systems deployed in hospitals. The tight integration with programming languages and maintainability attributes are good features to improve the development process. The database is well suited for smaller scale deployments of *openEHR* systems with a tight deadline and small maintenance budget. It should not be used where large sets of data such as laboratory need to be captured or intermediately stored. Db4o may be suitable for use in mobile healthcare environments which are synchronised with larger *openEHR* systems.

4.2 Intersystems Caché

Intersystems Caché is a post-relational object database powered by a multi-dimensional engine. The product offers a very different set of features to db4o. Data can be accessed in three different ways including an object interface, a modified version of SQL or multidimensional access. Using the multidimensional access is the most natural approach and may provide the best performance in

Caché. The object interface is essentially mapped to the multidimensional structures in the database engine. (Intersystems 2007)

4.2.1 Creating Classes in Caché

The features for creating the Caché database schema from the programming language class definitions are only available for Java using InterSystems Jalapeño. There are currently no native bindings for other programming languages supported by Caché such as .NET and C++. This project is most interested in using .NET and as such Caché classes cannot be created from the C# class definitions in the same way as dB4o. Instead, classes need to be created some other way. Caché supports XML and provides a facility to import XML schemas and convert them to Caché classes. This is a useful way of defining the classes in *openEHR* as all of the schemas corresponding to the specification are published (OceanInformatics 2006). Classes can also be defined in Caché studio or with Caché Roselink from Rational Rose UML models.

4.2.2 Accessing the database from C#

There are three main ways of accessing data in Caché from C#. ADO.NET or SOAP is a standard way of accessing data. However many of the benefits of the object-oriented approach are only available by using the Caché Managed Provider for .NET which projects Caché classes onto a .NET C# classes (Intersystems 2009e). Some issues were found with the Managed Provider which will be discussed in later sections.

4.2.3 Storing Objects in C#

The details of how to create the Caché Proxy classes do not need to be considered in this study in depth. FIGURE 14 shows how to create and save a proxy object. All of the proxy objects inherit an interface which is used for the persistence. If the *openEHRV1* classes are already defined in C# then a mapping is required where a proxy object is instantiated with the data members of the C# class. Alternatively it is possible to use the Caché classes directly to save objects and perhaps only use the mappings when retrieving objects. Unfortunately these problems make it difficult to achieve true transparent object persistence.

```
openEHRCache.EHR ehr = new openEHRCache.EHR(CacheConnect, system_id, ehr_id,
    time_created, ehr_access, ehr_status, directory, compositions,
    contributions);
// instantiate object members here ...
ehr.save();
ehr.close();
```

FIGURE 14 Saving a Caché proxy object in C#

4.2.4 Retrieving Objects in C#

The simplest way to retrieve an object from Caché in C# is to use the inherited method “openId(CacheConnect, id)”. This may be useful if the ID of an object to be retrieved is known - such as an entire EHR or composition. However *openEHR* maintains its own separate identification standard which differs from the object identification in Caché. As a result, most of the time a query will need to be generated. Queries can be made using ADO.NET or by encapsulating Caché’s modified SQL in the methods of the Caché classes. Alternatively, methods can be implemented in the Caché class which can manually search for the data needed in the object tree hierarchy.

4.2.5 Storage Capacity

A single database file in Caché has the potential to store much larger amounts of data than db4o. The theoretical maximum storage capacity is 32GB when the block size is set to 8kb. Practically this may not be achievable because of the Operating System limits on the number of files it can have opened at any one time. (Intersystems 2009d)

Caché is able to achieve higher volumes than db4o by automatically mounting the database files and providing the services required to manage several database files automatically. Using multiple database files with db4o requires the programmers to build a layer on top to manage a single point access to the entire collection of databases.

4.2.6 Maintenance

The most impeding limitation using Caché in terms of development time and maintenance is that true object persistence is not offered for .NET class definitions. However the initial mapping and maintenance required when changes occur is still significantly less than what is required for a relational or XML-relational approach.

4.2.7 Concurrency

Caché allows implicit locking to protect objects from potentially hazardous simultaneous access. Furthermore, clustering servers provides more resources to service requests and thus improve concurrency (Intersystems 2007). Db4o did not provide such a feature, although it is possible a custom solution could be combined with load balancing software.

4.2.8 Security and Encryption

Caché provides a more comprehensive set of security features than offered in db4o. In db4o access rights can be granted to one or more users for the entire database file which is specified as a call in the db4o API. This is a very minimalist approach in comparison to Caché which provides Role-Based Access Control (RBAC). Roles define the permissions for database resources and each user can be assigned to a set of roles.

Similar to db4o, Caché provides encryption to protect data from unauthorized disk access. However, Caché uses the Advanced Encryption Standard (AES) rather than XTEA. AES is another block cipher algorithm. Cryptanalysis shows it requires 2^{119} time complexity to break (Biryukov & Khovratovich 2009). In any case, the feasibility of breaking either is low. Both are fast and seem acceptable for healthcare, but recommendations for encryption algorithms are out of the scope of this study and should be addressed separately.

Caché also provides automatic audit logging for both system events and users access. However a separate auditing system is still required in *openEHR* as described in the specification.

Finally Caché provides Kerberos (MIT 2009) authentication which utilises a third party to authenticate a user and facilitate the authorisation of the user over an unprotected network. (Intersystems 2009c)

4.2.9 Distribution

Caché provides a feature called the Enterprise Cache Protocol (ECP) (Intersystems 2009b) which allows the database to be distributed over several heterogeneous servers. ECP has been designed so that applications programmers receive a logical view of namespaces which unifies access to the

distributed databases. This approach is primarily administrative and distribution is achieved by adjusting the runtime configuration rather than the databases themselves or application code.

Another feature which Caché provides that may assist with distribution is support for SOAP and Web Services. It is very likely that *openEHR* would be deployed in an SOA environment so that any application, can share information, even between systems using different databases and platforms. It is more likely that services in an *openEHR* system may be defined at a much higher level architecturally. However there may also be instances when a complete EHR needs to be transferred from one GP or Hospital to another and that can be achieved at the information level.

4.2.10 Fault Tolerance and Availability

Caché is designed for high availability so that when it is coupled with appropriate hardware configurations it can remain stable. It has significantly more robust features for managing fault tolerance and availability than dB4o and more in line with the requirements of health information systems. Unlike many databases, Caché implements an image journal of writes. Recovering the database from failure, results in completion of all updates that were transitional. This differs from most systems which roll back to previous states. This is extended with a global journal that can be configured to save particularly important updates since the last backup, which is likely to be at least several hours behind.

Another feature which Caché provides to assist with disaster recovery is shadow journaling and read-only reporting. Heterogeneous shadow servers running Caché continually obtain journal details from the production database and typically remain only a few transactions behind the production database system. Shadow servers connect to the production server over a network. (Intersystems 2009a)

In addition to shadow servers, Caché is able to manage hardware failures by allowing clustering. Multiple servers accessing the same disk storage can be configured as part of a cluster so that when hardware on one server fails, requests can be routed to the remaining servers in the clusters. (Intersystems 2009b)

4.2.11 Support

A comprehensive set of documentation is provided on all aspects of the product for both development and administration. There are few developers communities around to discuss the product and some issues found were difficult to work around. At the time of writing, Intersystems provides an optional world-wide support service.

4.2.12 Opportunities

The high availability and highly distributable properties make Caché a very possible choice for an enterprise health system. Scaling the database up doesn't necessarily require further programming but is, instead, included as an administration feature where nodes maybe added using the ECP.

Although no tight integration with .NET exists, the lower coupling between the database and the programming language and platform may provide other benefits. Mapping the Caché Proxy classes in .NET to the equivalent classes in .NET requires less effort and development time than what is required for a relational database.

The relational gateway is also a very useful feature which may help increase interoperability with legacy systems using relational databases.

4.3 Objectivity/DB

Objectivity/db is another object database that provides tighter integration with programming languages than Caché whilst providing more interoperability between different platforms than db4o. Support is currently available for C++, Java, Python and C#. Objectivity can be deployed in embedded applications, standalone applications on PC's or in client-server configurations. However Objectivity places much emphasis on larger distributed deployments of the database.

4.3.1 Storing Objects in C#

There are two ways that objects can be saved to an Objectivity database. Firstly, any class which requires persistence can extend `ooObj` (which is provided as a reference within a .dll). Another alternative is persistence by interface which helps assist with the issues of single inheritance and may provide a simpler way to persist existing *openEHR* Reference Model classes. Similar to db4o, the schema for classes are added to the database at runtime although when the application is started.

Understanding the persistence of objects to Objectivity/DB requires a basic understanding of how the database is organised. The top level container is the *federated* database which contains one or more *databases* which consists of one or more *containers* that group basic objects. Other databases mentioned do not have this notion of containers. The role of containers is to provide the minimal unit of locking and to help improve performance by segregating objects into logical groupings based on a range of factors such as frequency and ratio of reads/writes, location and availability. When an object is persisted it needs to be placed in a container in some database across the federation which is called *clustering* in the Objectivity documentation. This can be achieved either explicitly or implicitly by using a clustering strategy which finds the appropriate container for the object. By contrast db4o doesn't provide these distributed features and Caché ECP only puts objects in the local storage. (Objectivity, I 2006a)

FIGURE 15 shows how an EHR object can be stored in Objectivity/DB. The code is similar to both Caché and db4o except unlike Caché, the .NET classes can be converted to schemas and stored transparently. The session object provides the transaction mechanics for interaction with the database. The connection object must be established before opening sessions. In the C# implementation an instance of a class that implements the persistence interfaces is automatically made persistent when it is created within a session. (Objectivity, I 2008)

```
...
Connection connection = Connection.open(
    "myFD", // boot file
    oo.openReadWrite); // access level
Session session = connection.CreateSession();
session.BeginTransaction(OpenMode.Update);
EHR ehr = new EHR(system_id, ehr_id, time_created,
    ehr_access, ehr_status, directory, compositions, contributions);
session.CommitTransaction();
session.Dispose();
connection.close();
...
```

FIGURE 15 Storing a ooObj extended C# object to a default Objectivity cluster

Objectivity also provides features which optimise storage of particular objects such as collections. For instance, a scalable-collection of classes is provided that contribute to better performance for persistence than ordinary collections provided by other libraries such as the CLR. (Objectivity, I 2006a)

4.3.2 Retrieving Objects in C#

All objects in Objectivity/DB are assigned a unique identifier which can be used to identify it across the federation and the identifier for each object is known as its object identifier (OID).

There are several ways to find objects. Direct access to an object can be achievable by looking up the OID or object name or manually scanning for an object within a container or database by using an iterator. The method of scanning is similar to the normal method of iterating through a collection in object oriented programming languages. However, an additional feature is provided to scan based on predicates which are described as regular expressions on objects data members. The additional of predicates is similar to the WHERE clause in SQL.

A query optimisation technique that is available in Objectivity/DB not present in db4o and Caché is parallel queries. Queries may be run in parallel over several threads or servers to increase the speed of search. Of course these results in increased running costs as multiple servers are required to make the most of this feature. This is an example of an optimisation technique that could be applied to any database regardless of their database model. Such optimisation techniques clearly indicate that the performance is more closely coupled to the extent of the technologies implemented in the database management system and the developer's ability to make use of these optimisations when designing the system.

4.3.3 Storage Capacity

A single federation can contain up to 65,530 databases. The administration documentation (Objectivity, I 2006b) states a single database in the federation can store 256 TB with a total overhead of 88 GB (22bytes per page in 64-bit operating environment). Of course this is not a feasible maximum due to the constraints of the file system used by specific Operating Systems and the capacity of physical storage. For instance, the maximum file size in windows XP with NTFS is 16 TB and the practical maximum volume size of the complete file system as advocated by Microsoft is 2 TB (Microsoft 2009a). As a result, the maximum practical federation size would be smaller than the very large theoretical maximum. However, in comparison to Caché, the much larger maximum theoretical size may assist in providing the future-proof aspects *openEHR* advocates.

In real world scenarios implementations of Objectivity/DB have shown to manage over a petabyte of data such as at the Stanford Linear Accelerator Centre for BaBar (Becla & Wang 2005). This milestone was set by Objectivity/DB years ago may also be possible on some relational databases. For instance it was recently published that MySpace uses 440 SQL Server instances to manage 1 petabyte of data (Microsoft 2009b)

4.3.4 Maintenance

Since the Objectivity/DB schema is created from internal class definitions from the programming language, it is easier to maintain changes to the schema. This is more like the features provided in db4o as no mappings need to be maintained between schemas. In addition to the basic features that db4o implement, Objectivity provides access control to prevent certain applications from updating or changing the schema. Due to the constraint-based nature of archetypes it may be rare that new classes need to be added. It is quite possible a revision or alternative to the *openEHR* domain model could remove the reference to archetypes from data structures and instead automatically map archetypes to classes which can make use of the schema management for persistence. Although this may blur the clean separation of knowledge and information and increase complexity so it is unlikely this scenario will occur. It would also complicate the versioning of archetypes.

Objectivity/DB also provides a feature for supporting changes to the schema. Ideally changes should be phased in and not adjusted at runtime to ensure consistency across the application and other applications using different programming languages. However this feature in addition to object conversion allows the schema to be changed quite easily and automatically update the previously stored objects to be compatible with the new schema. There may be issues with this automatic conversion where references or data is removed from new versions which may still be needed after the change occurs. This is particularly a problem with *openEHR* as the data from old versions should still be complete.

4.3.5 Concurrency

Unlike some other databases which require separate connection objects for each application Objectivity/DB only requires a single Connection instance for the entire application. Instead Sessions are used to safely interact with the federated database and they can be pooled quite easily to allow concurrent access from a single application.

Locking in Objectivity/DB is not performed at a fine-grained level on objects. Instead the minimal unit of locking is performed on containers which are discussed in distribution section. Both implicit and explicit locking is provided, but the course-grained unit of locking may present issues in writing many new *openEHR* versioned compositions simultaneously. Careful design and partitioning of the reference model into appropriate containers would be needed to ensure effective implementation. Furthermore propagation of locks to composite objects requires establishment of a reference linking system which requires more effort.

Similar to Caché, Objectivity/DB also provides a high availability package which assists not only in fault tolerance but support for many users accessing the system concurrently from various locations. (Objectivity, I 2006a)

4.3.6 Security and Encryption

A significant drawback seems to be that Objectivity/DB relies completely on the operating system and file systems for access control. There is no role-based or discretionary access control in place. A subsequent layer would be needed to implement the details of the *openEHR* access control.

Objectivity also does not appear to provide any inbuilt encryption so it is another feature that would need to be built into a middle layer to service the requirements of the *openEHR* specification.

4.3.7 Distribution

Although Objectivity can be embedded, deployed on the same host system or used in a client-server environment, it is quite clear that it has been specifically designed for highly distributed systems. This is quite clear by the logical organisation of the objects in the database particularly with the top level federation database. The federated database contains the overall schema, a catalogue of database and locations, indexes, journals, lock management and locations of systems resources in a boot file. Databases in the catalogue can be placed on the same host system or distributed across many nodes.

The idea is similar to Caché ECP but Objectivity provides a more fine-grained control over the way the database can be distributed. This may or may not be an advantage depending on the deployment and project characteristics. Clustering strategies may assist in making a distribution more

transparent for the usual application programmer. Further enhancements in comparison to Caché include parallel queries where a query can be distributed over several query servers and threads. (Objectivity, I 2006a)

Studies have been conducted that attempt to find the best method of integrating Objectivity/DB in a SOA environment. For instance GridwiseTech (2007) suggest using OIDs as the parameters of web services to remove the overhead of SOAP. They also suggest using SOA as a layer providing networked access to the methods of the objects that are stored as data in Objectivity.

4.3.8 Fault Tolerance and Availability

Unlike Caché, Objectivity/DB provides its high availability package as an additional product. Nevertheless it provides useful features which assist in improving fault tolerance and availability.

Similar to Caché, journaling is supported, but by default this is set to roll backward rather than forward. The manual alludes to the possibility of forward recovery without mentioning how this is possible.

Partitioning of the database is achievable using the high availability package. Each partition groups a set of databases and provides its own resources for managing locks and journals thus increasing the availability. Images or copies of these partitions can be replicated whilst still providing a single logical entry point for transparent access. This functionality is similar to the shadow server functionality that Caché provides although more than two images can be maintained and unlike clustering in Caché they do not need to share the same storage resources. (Objectivity, I 2006c)

4.3.9 Support

Representatives from Objectivity were quite willing to discuss the product. Furthermore the documentation provided by Objectivity is rather comprehensive and very complete. For instance, the user guides just for the Java bindings is close to 1000 pages and significant documentation exists for each binding and administrative issues. Similar to the Caché learning tools, Objectivity also provides a web based training website (Objectivity, I 2009). The only drawback is that learning tools for .NET and the web-based training seems to still be in development. A reasonably comprehensive API is available for C# in the mean time as more support for C# is added.

4.3.10 Opportunities

Objectivity/DB is perhaps the most scalable database reviewed with the ability of many fine-grained control features to optimise performance. This includes replication to reduce network congestion and the possible size for the database. Distribution transparency can be obtained through the use of clustering strategies, but may be harder to implement and maintain than the features provided by the ECP in Caché. The tight integration and schema evolution features make it easier to maintain in other aspects. An Objectivity/DB implementation integrated with SOA as described in GridwiseTech (2007) is an extremely viable option for very large scale deployments in large hospitals or networks of health providers. Both Caché and Objectivity/DB provide the high availability features required for healthcare but the best choice depends on the scale of deployment and budget constraints.

5 Preliminary Evaluation

It is very difficult to compare an implementation of a database system upon the access model alone. The underlying technologies and effectiveness of implementation may result in drastically different scalability and performance between database systems using the same access model.

This section further investigates the possible issues and potential opportunities of Db4o and Intersystem's Caché by implementing a simple linear recursive object model. This object model has some of the characteristics of an *openEHR* Reference Model. The aim is to select a database system to use for the final evaluation and elicit some possible implementation strategies. Unfortunately, the simple object model and the *openEHR* Reference Model was not implemented in Objectivity/DB due to the short time frames allowed for trial. However, Objectivity/DB will be discussed and compared in later sections as the structures it provides may be the best way of realising the best implementation approach found from the preliminary evaluation.

The preliminary and final evaluations are strictly focussed on the performance characteristics of different implementation strategies in different technologies. However, other important characteristics should be considered during the requirements phase of an *openEHR* project such as the Total Cost of Ownership, Security, Reliability, Concurrency and Maintainability. Some of these characteristics were considered and discussed in SECTION 4.

5.1 Testing Environment

All tests performed in the preliminary evaluation were conducted in a controlled environment on the same system so that no data needed to be sent across a network. This helped to provide an indication of the performance of the OO implementation features in comparison to the XML-Relational mapping approach. However, as each database handles distribution and networked caching differently and future research may expand into testing distributed queries, parallel queries and concurrent loads.

TABLE 3 displays some basic information such as the Operating System, Hardware and Software used in the system that was used during the preliminary evaluation.

Operating System	Microsoft Windows XP Professional x64 Edition SP2
Motherboard	TYAN K8WE (S2895)
Processor	2 x Dual Core AMD Opteron™ 270 Processors (4 cores total at 2.01 Ghz each)
Memory	8190MB DDR1 400mhz ECC Registered (NUMA)
Hard Drive	Western Digital WD5000AAKB 500GB
.NET Framework	3.5
Db4o Version	7.4 for .NET 3.5
Intersystem's Caché Version	2008.1

TABLE 3 Relevant system hardware and software specifications for testing environment

TABLE 4 shows the specifications of the hard drive published by the manufacturer.

Manufacturer	Western Digital
Model	WD5000AAKB
Formatted Capacity	500, 107 MB
Interface	SATA
Average Latency	4.2ms
Buffer	16 MB
Data Transfer Rate (buffer to disk)	70MB/s sustained
Data Transfer Rate (buffer to host)	3 Gb/s maximum
Rotational Speed	7200 RPM
Start/stop cycles	50, 000

TABLE 4 Western Digital WD5000AAKB Hard Drive specifications for the preliminary evaluations
(WesternDigital 2008)

FIGURE 16 displays the benchmark results collected from 'HD Tune 2.55' with the default setup (EFDSsoftware 2008). These results were useful for analysing the overhead associated with each database during insertion and retrieval during activation.

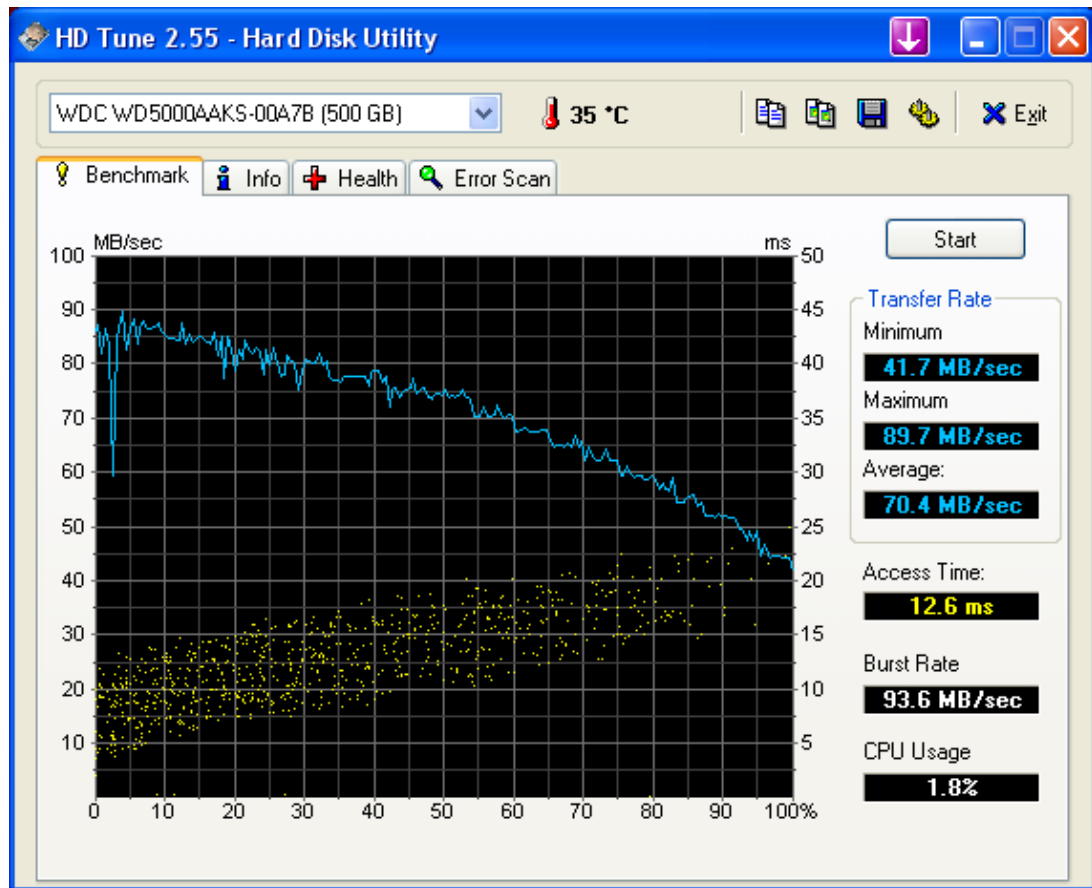


FIGURE 16 Results from 'HD Tune' Benchmark for the Western Digital Hard Drive

5.2 Measurement Toolkit

Recording the time elapsed for a particular transaction or collective set of transactions is a frequent standard form of measurement used for gauging the performance of a database. This is used in studies such as Objectivity & Violin (2008), Zyl, Kourie & Boake (2006), Austin (2004), Schaller (1999) and Ohnemus (1996). However, the actual measurements taken differ immensely depending on the application area or benchmark criteria.

For the purpose of this study, all tests were performed within the .NET framework using C# as the client side programming language. For consistency, the time elapsed between blocks of code was accurately returned using the `System.Diagnostics.Stopwatch` class provided by the .NET framework. A property exists to retrieve the ticks elapsed between calling `Start()` and `Stop()`. Retrieving the timer frequency allows us to calculate the time in 100th of a nanosecond by dividing the ticks by the frequency.

A single measurement on its own is not very useful due to the variation of system resource utilisation over time. Another factor is that secondary operations. These are quicker due to most database systems implementing some form of caching features. A measurement toolkit was developed to overcome these problems. The results of a single test can be added to a group of logical tests and logged for further statistical analysis. The toolkit is extensible so that different types of loggers may be added. Currently the test results can be stored in a db4o database and/or as human readable text. See APPENDIX A for more details.

5.3 Object Model for Initial Evaluation

FIGURE 17 displays the class diagram used for the preliminary testing. It is essentially a linked list providing a recursive structure. This structure was chosen because many of the sub-trees in a COMPOSITION in *openEHR* allow objects to refer to themselves. This does not test the component type of structures existing in the *openEHR* content items. However, the primary aim of the preliminary tests was to focus on the implementation details and general aspects of each object database. Full implementations of the *openEHR* object model are considered later.

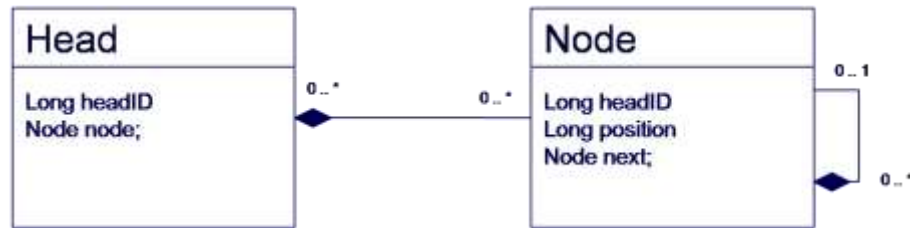


FIGURE 17 Linear Recursive Structure used for Preliminary Testing

5.4 Implementation strategies

5.4.1 Db4o implementation

Originally there was no intention to place the 'headID' in the node object in FIGURE 17, so that all nodes could be reached by narrowing the scope of the query or traversal. However, this introduced several issues in db4o. Firstly, the query mechanisms does not allow retrieval of only smaller objects composing the top-level object specified in the query. Translating the problem to an *openEHR* structure requires a query such as that described in FIGURE 12. The issue with such a query is that the whole EHR, including every COMPOSTION, needs to be retrieved in order to find one instance. Retrieving the whole composition may compromise performance in content based queries.

The alternative was to store the ancestor objects identification field in node objects, (which may correspond to "ELEMENT" objects in *openEHR*) and perform the query on the object which the application will bring into memory. For db4o this seemed to be the most appropriate approach although other structures such as hash tables were considered. Unfortunately performance was inadequate using the .NET structures for lookup, probably because the hash table was not being cached on the client side. The test was run on the exact structure shown above, but as two sets of tests, one with just an index on "headID" and another with an index on both "headID" and "position" in a Node. Performance without any indexes, as expected, performed drastically worse on read operations and is not included further in this comparison.

5.4.2 Intersystem's Caché implementation

Caché offered several more implementation options than db4o including: dynamic SQL queries, path traversal, global traversal (Intersystems, 2007) and finding objects by a unique object ID. Globals are a concept in Caché that is used to describe multi-dimensional structures. A global node

corresponds to a single persistent multi-dimensional array. These arrays can be indexed which can be used to find global nodes directly or to traverse on certain ordered indexes. It is important to note that the node in the object model presented above is not related to the notion of a global node.

All three options were evaluated. From an implementation perspective, using queries should be the simplest to implement. However dynamic queries are the only way of performing queries prepared at runtime. As a limitation, the paths in SQL query strings can only be 255 characters long. Storing ancestor information in the Node like in the db4o approach above may help confine the limit of the query length and be the only option for using queries in Caché.

The traversal approach required loops and control flow statements to traverse the object graph. In theory, this is ideal because the "EHR" object can firstly be retrieved by lookup narrowing the scope of the search. Since the paths provide the navigation to the data, object references can be traversed removing the need to search and only requiring compare operations. However, in practise, the traversal using loops and control flow provided a rather large overhead to items that are very deep in the object tree. Essentially, performance declines as the search traverses deeper nodes. Performance becomes even worse if the contents of each object are transferred to memory.

The final two options using globals or object identifiers are both similar. In the same way that indexes can be allocated to globals, object tree path information can be placed in the string identifying the object. Being able to use the path information to find objects in *openEHR* is ideal because of the notion of the abstract LOCATABLE type. Many of the types in the *openEHR* RM are derived from the LOCATABLE type, which uniquely identify these objects within the extent of an archetype. Since traversal is slow, indexing path information provides a faster alternative to find not only COMPOSITION objects but also content items that are deeper in the object hierarchy.

Mapping AQL queries used in *openEHR*, such as the example shown in FIGURE 12 is possible. The "SELECT" section of the query defines the paths to locate and return. Combining the archetype node ID, EHR ID, and Archetype ID provides enough path information to return a list of corresponding COMPOSITION instances using global index traversal or iterating over object ID's. The "WHILE" part of the query can be evaluated in languages such as C# using delegate functions, similar approach to the Native Queries db4o uses (See *Section 4.2*).

From all the approaches, the direct lookup approach performed the best in this initial evaluation. Furthermore, it can be maintained as a separate structure since the mappings are for object IDs rather than references. This helps to maintain modularity and separation from the *openEHR* reference model specifications. However, similar to paths in Caché queries, in practise this approach may be limited by the length of the global and object ID strings in Caché. Although Objectivity/DB is not experimentally evaluated in this study, the "ooCollections" structures and name-based lookup approaches provided by the product may produce even better results.

In all of the above global and object ID scenarios, it is possible to also use the indexes to refer directly to the node or object rather than an ID key.

5.4.3 Summary of Test Configurations

Where possible each configuration uses the best approach found. For instance QbE, Native Queries and different orderings of the SODA queries were tried before finding the approach used for comparison. The summary of these preliminary test configurations are displayed in TABLE 5.

Name:	Approach
Caché Query	Dynamic query on the node objects Indexes on: head.headID, Node.headID, Node.position
Caché Traversal	Traversing references according to the path details specified
Caché Globals	Using the multi-dimensional features of Caché to retrieve nodes. A separate class was created to store all the path information as an index and the object ID of the node at that position as a value that can be retrieved from the global
Caché Lookup Ref	Provides a single combined index on the path details as a string which separates path information with a "-" character. The index is set as the key to the object so there is no intermediate object ID lookup.
Caché Lookup OID	Provides the same approach combined key approach as above but stored in a separate class which also stores the object ID.
Db4o Query N	SODA Query - No index on Node.position Indexes on: head.headID, Node.headID
Db4o Query I	SODA Query - Extra index on Node.position Indexes on: head.headID, Node.headID, Node.position

TABLE 5 Preliminary Test Configurations used to evaluate OODB's implementation features Results

5.4.4 Bulk Insertion Time

Operations: Insert 10,000 head objects containing 100 nodes each.

There is an obvious overhead for any of the configurations that require additional lookup structures. This could be removed for the globals configuration. The results shown are quite similar.

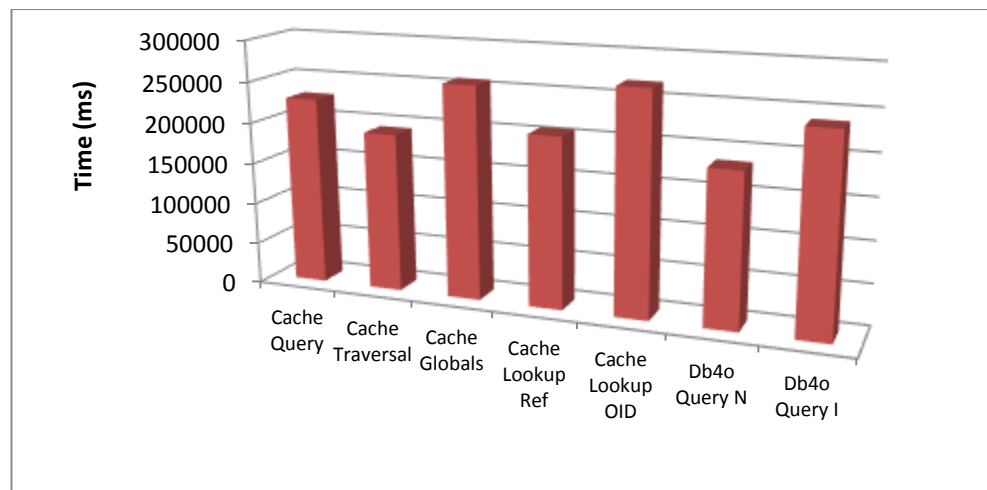


FIGURE 18 Preliminary Evaluation: Bulk Insertion Time

5.4.5 Insertion at Fixed Intervals

Number of Iterations: 50

Operations: Insert 1,000 head objects followed by 50 head objects (measuring time to commit the 50 - one at a time)

It is more likely that entries in *openEHR* will occur one sub tree at a time rather than in many at a time. This test determines how the time of insertion is affected as the database size grows.

FIGURE 19 shows the results of all configurations. The performance of Db4o typically declines as the size of the database increases particularly for the Db4o I configuration.

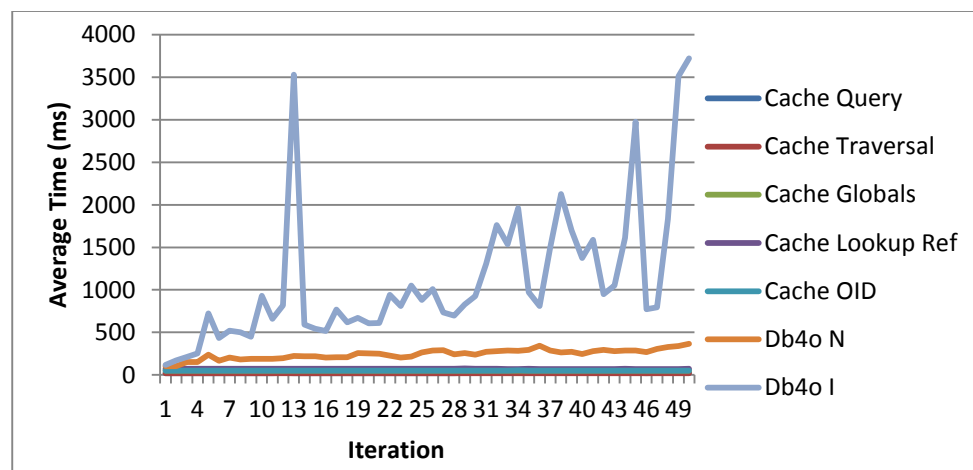


FIGURE 19 Preliminary Evaluation: Insertion at Fixed Intervals (Caché and Db4o)

FIGURE 20 shows only the results from the Caché configurations. In comparison to the Db4o configurations all are much faster and performance does not degrade as the size of the database increases. The higher insertion time for the lookup methods can be attributed to the fact that double the amount of objects are stored in a separate transaction.

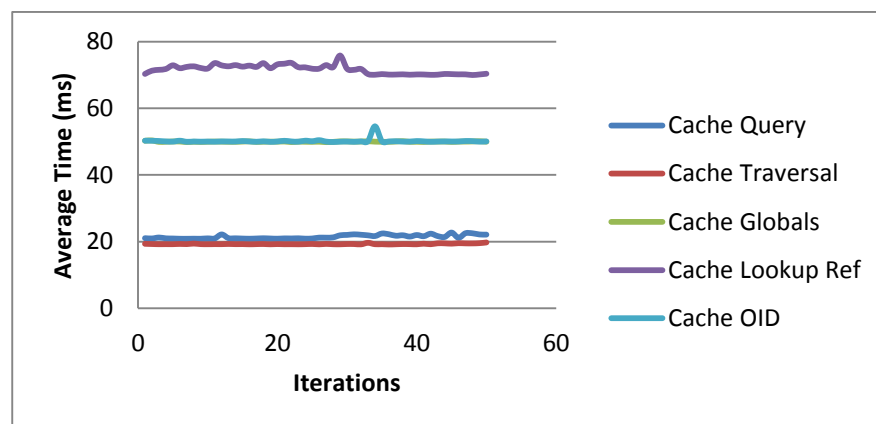


FIGURE 20 Preliminary Evaluation: Insertion at Fixed Intervals (Caché only)

5.4.6 Find different sized nodes

Number of Objects: 1 head object each with 50 nodes

Number of Iterations: 10

Operation: Find each node in the only head object

The data structure has been extended so that every 10th node has an array of 8,000 nodes. Ideally this could have been much bigger for the test, but was minimised due to some issues persisting arrays in Caché. Firstly, multidimensional properties of classes are only transient. Secondly, List and Array types would only store a certain amount before encountering unreadable errors. The only other alternative is to store the arrays as separate globals. However this removes some of the Object-Oriented properties. The minimal amount of list objects that can be saved should be sufficient for most *openEHR* paths though.

The purpose of this test was to ensure that adding large nodes does not affect the time to find nodes without arrays.

The actual arrays are not activated in both Caché and Db4o. However there is a small overhead for finding the larger nodes in most techniques to varying degrees. Db4o N was removed as it had a similar pattern but the average time was much larger overall.

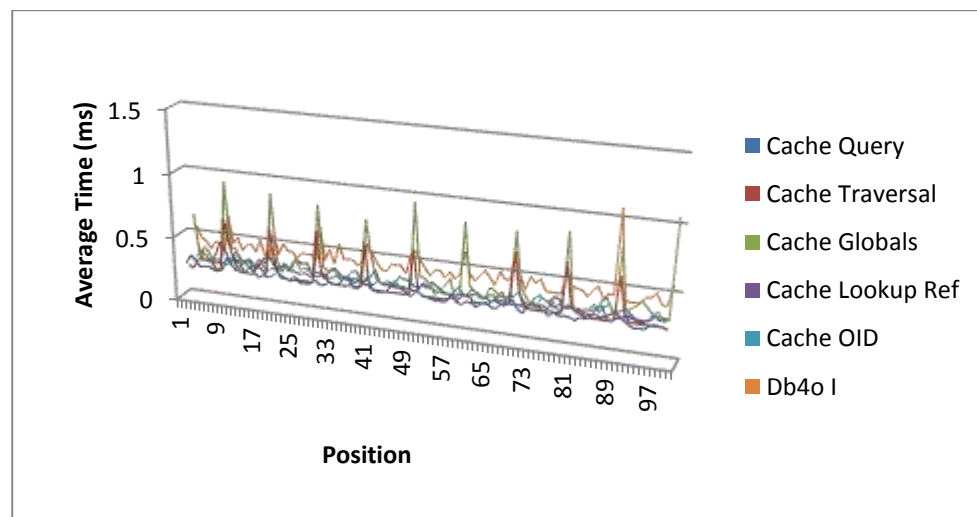


FIGURE 21 Preliminary Evaluation: Find Different sized nodes

5.4.7 Find Single Node

For both tests: number of head objects: 5000 (blue), 10,000 (orange). All head objects contain 100 nodes each.

NON-CACHED RESULTS

Number of times ran: 5

Number of Iterations: 1

Operations: Find node at position #50 within a head object with hID = 2500

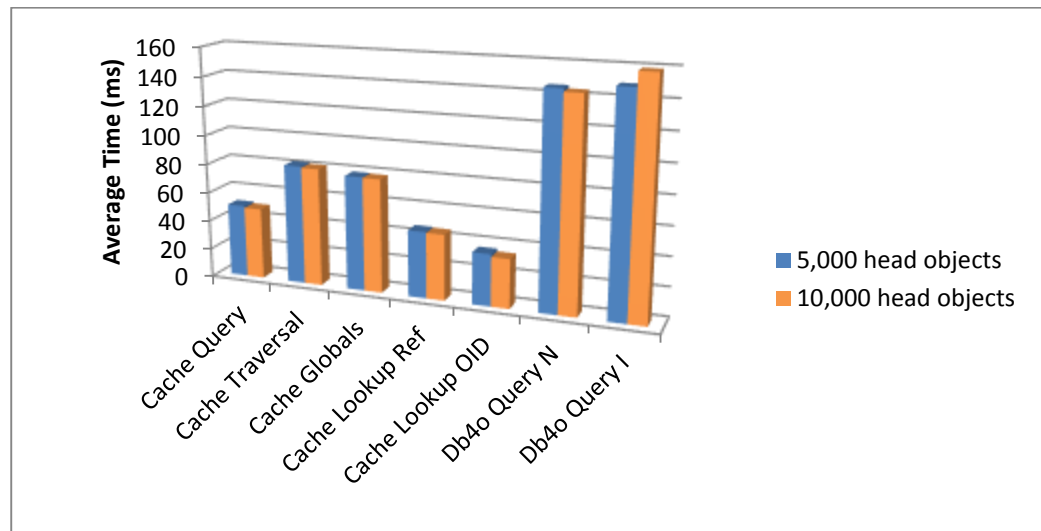


FIGURE 22 Preliminary Evaluation: Find a single node (Non-Cached Results)

CACHED RESULTS

Number of times ran: 1

Number of Iterations: 999

Operations: Find node at position #50 within a head object from hID = 2000 to hID = 2999

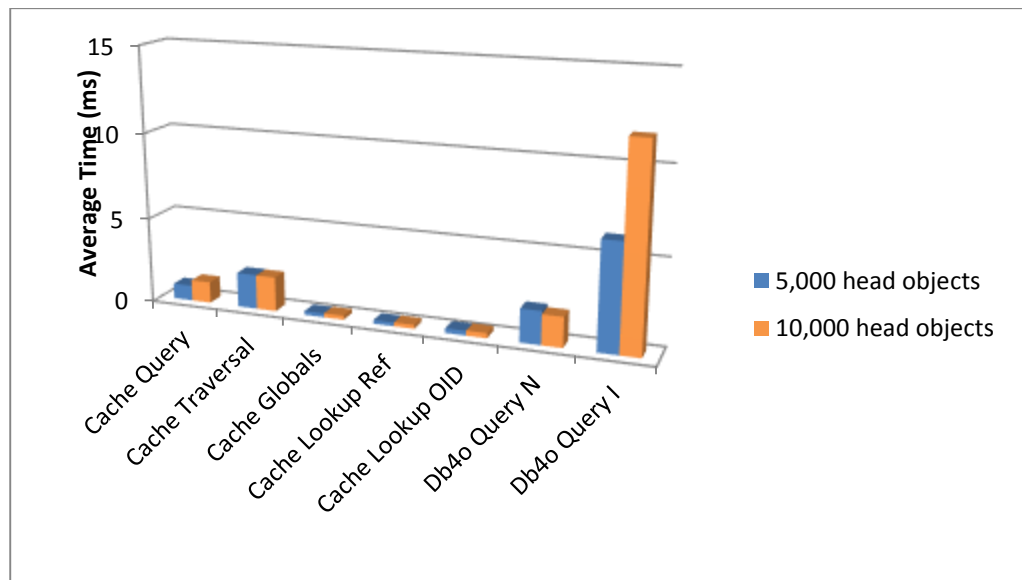


FIGURE 23 Preliminary Evaluation: Find a single node (Cached Results)

5.4.8 Find Group

Number of times ran: 6 times with gap sizes: 5,10,20,50,100,200 and 10,000 head objects total

Number of Iterations: 999 (only included cached results)

Operations: Find all nodes inside head objects with identifiers in between 2500 and 2500+gap size

FIGURE 24 shows the results of the above test for all configurations. Initially the performance of the Db4o configuration N performs better for smaller numbers of objects, but Db4o I begins to perform better at around about 100 head objects. Whilst the Caché traversal configuration mostly performed better than the db4o queries, it may not be the case if the position of the node was closer to the total number of nodes in each head object as this adds further traversals. Caché queries performed significantly better than db4o in this test; however all of the lookup techniques perform considerably better than every other configuration.

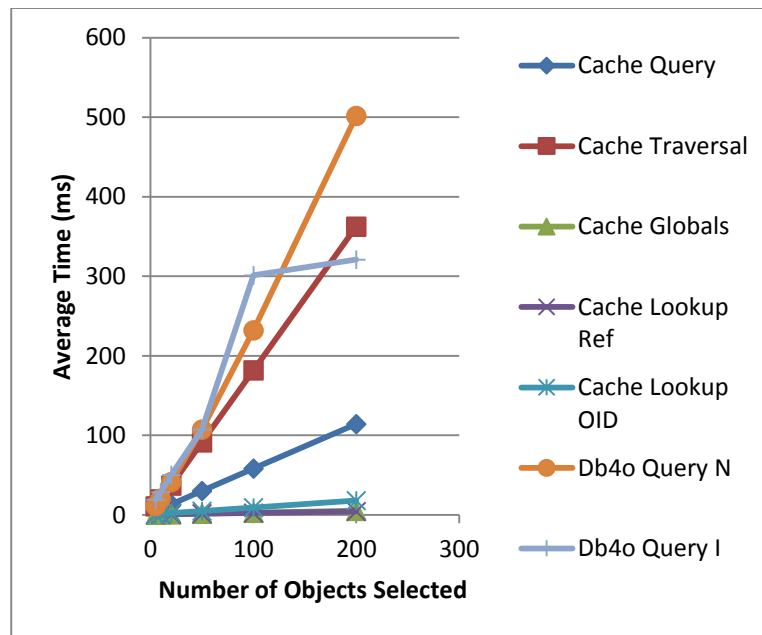


FIGURE 24 Preliminary Evaluation: Find groups of nodes with in specified ranges

FIGURE 25 examines the lookup techniques more closely. The Caché Globals and Lookup Ref provide the best performance with the average time taken being directly proportional to the number of objects.

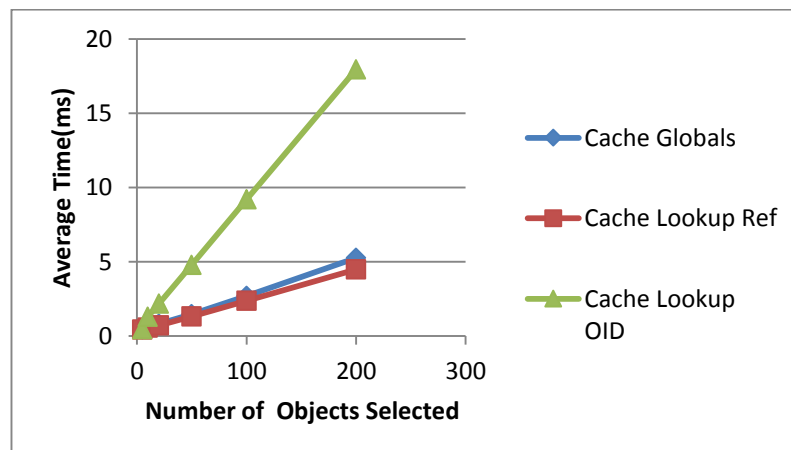


FIGURE 25 Preliminary Evaluation: Find groups of nodes (fewer configurations)

5.5 Preliminary Evaluation Summary

The results of the preliminary evaluation show how critical the implementation techniques and choice of Object-Oriented databases are in order to achieve outstanding performance. These results mostly focused on the performance of finding the objects rather than retrieving a reasonable portion of data

The preliminary results have shown that the traversal technique performed worse than every other technique except for the Db4o I configuration. However, traversal of larger object trees is very likely to provide even worse performance than the Db4o configuration. This feature of OODBMSs is not available in RDBMSs.

In general, the Caché implementations outperformed the Db4o implementations in every test. It may be possible to achieve better performance closer to the results in the Caché implementations by using some of the structures available in the Db4o API and using a direct lookup based approach.

All three lookup techniques in Caché provided better performance than queries except for the globals implementation for finding a single node before caching. Using path information as a key to the data is also a natural approach for *openEHR* since AQL queries are path based.

A few issues may need to be resolved in order to make the lookup technique feasible in a large scale deployment of an *openEHR* systems. The limited global substring length in Caché may be an issue. Other OO databases such as Objectivity may not have this problem. For instance an object can be set as the key to another object. As such, all path information could be expressed in the key object. Other collections objects are also provided, although the fastest ooHashMap can only scale to about 10,000 objects (Objectivity, I 2006a). However, in general this lookup technique would probably benefit from being split into different scopes based on the demographic location of the objects and the extent of the object tree.

6 Final Evaluation

This section evaluates and compares an existing XML-Relational based approach for the *openEHR* persistence layer with an Object-Oriented based persistence layer prototype. It also discusses the implementation of the persistence framework in Caché and the issues identified. The existing *openEHR* persistence layer has been supplied by Ocean Informatics and uses a Microsoft SQL Server (MS SQL) 2005 and stores Compositions with the assistance of the Fast Infoset (FI) Standard and ASN1 (Sandoz, Triglia & Pericas-Geertsen 2004). Some tests were also performed on the Ocean Informatics previous implementation using the MS SQL Server's Native XML binding previously discussed in SECTION 3.2.

By contrast, the Object-Oriented based persistence layer prototype has been developed specifically for this research and uses an Intersystem's Caché database implemented with the best performing techniques discovered from the previous section. The MS SQL implementations will not be discussed in detail as they are used in Ocean Informatics' proprietary products. It must be made explicitly clear that the comparison is based on the implementation and not aimed at comparing the products themselves. The database products may perform differently when used in other applications and changes to the implementation approach will indubitably influence the results.

Some implementation issues were encountered, during the evaluation. The composition queries (discussed in SECTION 6.1) could not be measured due to these issues which are also discussed in SECTION 6.1.2 and APPENDIX C. Originally only Intersystem's Caché was planned to be used for the final evaluation based on the results in the preliminary evaluation (SECTION 5). However, db4o was used in place of Caché for the tests which could not be performed using the Caché based implementation of the *openEHR* persistence layer.

6.1 Prototype and Implementation Considerations

The partial prototype has been developed in conformance with the *openEHR* standard Release 1.0.1 Reference Model. All data in the reference model can be expressed in the prototype, although it was decided to only implement a reduced set of functionality and constraints since the test data will

already be structurally valid. The missing functionality and constraints are any invariants and functions that were not required to complete the testing and implement the basic use cases.

The prototype makes use of a set of XML schemas which define the data structures in the *openEHR* RM. This set of XML schemas are used to C# .NET class definitions by using the “xsd.exe” tool distributed by Microsoft (2009c). The schemas were also used to generate the corresponding class definitions in Caché. The reason for doing this was to ensure that the two systems are interoperable, which is one of the key aims of the *openEHR* foundation.

The client end of the prototype was programmed in C# in conjunction with the .NET Managed provider for Caché. An assembly of Caché Proxy classes were generated for C# in order to interact with the Caché database. The classes provide a complete representation of the types on the Caché database with additional features to persist and retrieve objects. They can arguably be used without any additional higher level .NET layers. However, there were several reasons to maintain a separate data model in C#. The persistence interface provides potential operations that higher levels in the *openEHR* architecture do not allow such as updating objects (due to versioning). The proxy objects are also potentially fragile as they require a connection to the database which may be dropped during network interruptions. Finally, in their current state they are not compatible with XML serialisation and de-serialisation, which were required to import the test data into the C# objects.

Since the test data was defined as XML files, these needed to be de-serialised into the *OpenEhr.V1.Its.XML* package and then converted into Caché Proxy objects in order to persist them. Conversion was achieved by creating a program that automatically generates a static class “RMConverter”. The generated class provides overloaded methods to convert any type from the .NET data types and the *OpenEhr.V1.Its.XML* to and from the Caché Proxy objects. The *OpenEhr.V1.Its.XML.RM* object instances were used as the parameters in the Facade encapsulating the core functionality required by the persistence layer defined by the use cases. The details of the generation of RM classes and conversion classes are displayed in FIGURE 26.

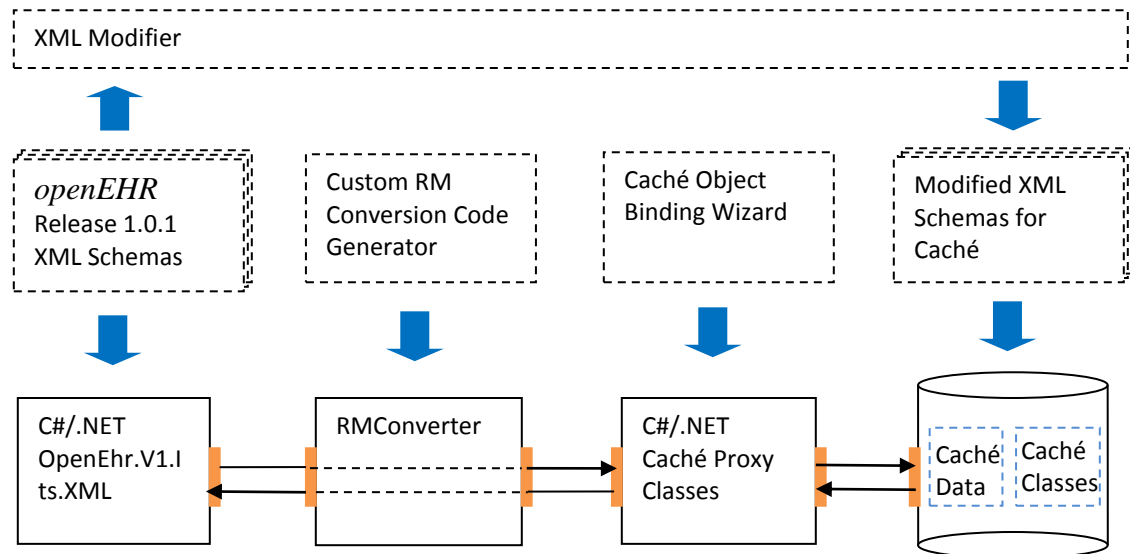


FIGURE 26 Generation of RM Classes, Caché Classes, Proxy Classes and Conversion facilities

6.1.1 Persistence Layer Requirements and Use Cases

As *openEHR* systems are layered, a well-defined set of requirements and use cases were elicited to reduce the scope of the persistence layer to its core functionality. This set provided the basis for the granularity of the test operations and performance which are discussed later. The key requirements and use cases can be summarised below in less formal terms:

1. Create EHR

Store the EHR with a unique ID, associated properties for controlling access, status and an empty list of contributions and versioned compositions.

2. Commit COMPOSITION instances

Store one or more COMPOSITION instances as a VERSION, belonging to a VERSIONED_OBJECT in an EHR.

3. Retrieve meta-data for a particular COMPOSITION based on various selection criteria

Data retrieved shall include an OBJECT_ID and all other meta-data for a particular set of health events, for example laboratory reports or Hospital discharge summaries

4. Retrieve a COMPOSITION by an OBJECT_ID

This feature allows one to directly retrieve the object after selecting it from a health event list produced in the above function

5. Find COMPOSITION objects corresponding to a particular ARCHETYPE_ID

This function shall retrieve all COMPOSITIONs for a particular patient (or any patient for a population query) which is a specific type of composition dictated by the composition level archetype

6. Find content at a particular node within an archetype

This function shall retrieve an archetype node at a specified path within an Archetype for all compositions containing that archetype.

Constraints such as values existing in the WHERE clause of AQL queries do not necessarily need to be evaluated at the persistence layer as this would be included in the query engine.

6.1.2 Global, Object Reference and Query Techniques

Continuing from the findings in *section 5*, an approach based around using path information as the key to archetype nodes within the content of a COMPOSITION was used for the Caché prototype.

FIGURE 27 shows a possible format which can be used as the basis of Caché globals or object identifier keys. All information can be elicited from the *openEHR* structures although a label will be required for each archetype within a particular COMPOSITION since each COMPOSITION can contain duplicates of the same archetype structures at different node levels.

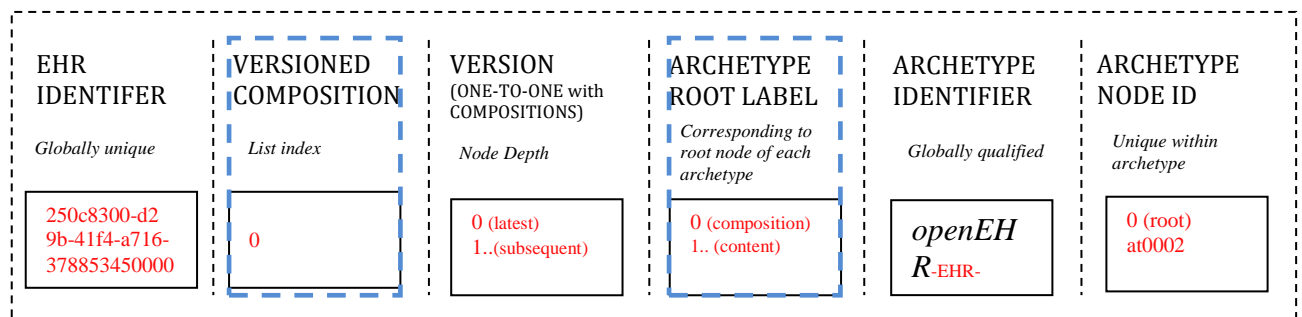


FIGURE 27 Contextual information that can be used to express paths as keys to object identifiers

The blue areas in FIGURE 27 would usually be omitted from the Caché global retrieval to obtain all object identifiers of the archetype node ID for all archetypes within all versioned compositions for a particular patient. Optionally the EHR ID can also be omitted for population queries.

Some potential issues were identified with using the information provided in FIGURE 27. Firstly, since there is no definition of an "archetype root label", further processing may be required to

generate labels for them. One such approach could be a tree search, which traverses all the paths within the content of a COMPOSITION as it is inserted into the database. This could be similar to the overhead of indexing relational databases but may potentially provide even worse insertion performance.

One solution to the lack of identifiers for the same archetypes within the context of a COMPOSITION is that the future *openEHR* specification can be updated to accommodate this. Providing similar sharable definitions of templates might solve this. Essentially the archetypes in a template can be labelled similar to how nodes are labelled within the context of an archetype. It may be beneficial to include extra ancestor information within any LOCATABLE object in addition to the archetype id and node id. This may include the EHR ID and VERSION ID as discussed in *section 5.4.2*. The result of this is that the extra contextual information about the containment of an archetype node can be added as the objects after validation rather than at the persistence level.

If it is desirable to not only find archetype nodes within the content of COMPOSITION but rather within any LOCATABLE item, the structure in FIGURE 28 can be used. Other structures that can be archetyped and versioned include EHR_STATUS and EHR_ACCESS which are attributes of the EHR rather than a CONTENT_ITEM of a COMPOSITION.

FIGURE 28 also provides some other solutions to the potential issues in FIGURE 27. For instance FIGURE 27 was implementation specific using the array indexes for the values of the VERSIONED_COMPOSITION and VERSION indexes. By contrast FIGURE 28 uses all unique identifier for these values. The obvious drawback is that the overall substring may be too long for the substrings in Caché. It may be possible to reduce the overall substring size to the allowed limit by changing VERSIONED_OBJECT<T> to a LATEST_VERSION field which has the value of 1 or 0 to indicate true or false. In fact, if globals are used instead of object keys, archived versions could be moved to a separate global structure.

Finally, another alternative that FIGURE 28 provides to FIGURE 27 is no need to traverse object graphs to label archetype roots. Instead an archetype number can indicate which archetype the node belongs in satisfying the current *openEHR* specification. However, this approach makes it difficult

to determine where the node is located in the hierarchy and in future may not be the most desirable approach even if it is currently valid.

EHR IDENTIFIER	VERSIONED_OBJECT<T> (or latest_version)	ORIGINAL_VERSION	ARCHETYPE IDENTIFIER	ARCHETYPE NODE ID	ARCHETYPE CONTAINMENT
<i>ehr id</i>	<i>uid or true/false</i>	<i>uid</i>	<i>Globally qualified</i>	<i>Unique within archetype</i>	<i>Unique within containment structure</i>
250c8300-d2 9b-41f4-a716- 378853450000	ab679eae-6074- 4871-8dda-561 8bbdbc210::7259 38E9-1C61-4C8 8-9BC62FC6 8731CFCB::1 OR TRUE/FALSE	3056c591-a4af- 725d8d1d 0652a 5d23459::72593 8E9-1C61-4C82- 6AE3-2AB6873 1CFCB::12FC68 731BCAB::1	<i>openEHR</i> EHR- INSTRUCTION. referral.v1	0 (root) at0002	0..*

FIGURE 28 Information to lookup any archetype node within LOCATABLE derived objects

Another obvious question regarding the performance of the structures used in FIGURE 27 and FIGURE 28 as globals is whether or not the Caché database engine can handle very large numbers of global nodes without reducing performance. There will almost certainly be tens of billions of nodes in a database after 5 or so years of use in a large health information system. However, papers such as one by Holtman and Stockinger (2000) indicates that it is certainly possible to have lookup structures that scale to 10 billion objects without much loss of performance.

Given the concern over the scalability of globals when the number of nodes becomes large, further investigation was required. To evaluate this premise in Caché's storage, a global in the format "^globalTEST(e,c,a,n)" which mimics a simplified version of FIGURE 27 was tested. The code used to store the globals is shown in FIGURE 29.

```

ClassMethod insert (start As %Integer, end As %Integer) {
    for e=start:1:end {
        for c=1:1:50 {
            for a=1:1:5 {
                for n=1:1:20 {
                    SET ^globalTEST(e,c,a,n) = e_c_a_n
                }
            }
        }
    }
}

```

FIGURE 29 Code to store a global structure using the initial single structure approach

The method "insert" was called 200 times from C# proxy classes with intervals of "e" set to 100. The results of each test instance was timed and collated into FIGURE 30 to show the change in the

index write time as the index grows. The total database size just for the indexes after inserting 500 million global nodes was 8679mb. There is no significant effect on the index write time as the number of globals become large.

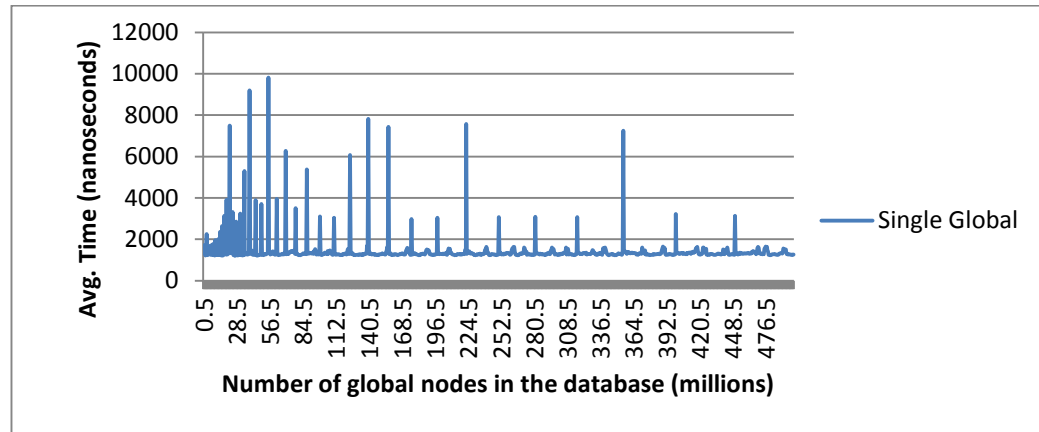


FIGURE 30 Average time it takes to insert a single global node into the database as the time grows

The average ms/read for a single global node was 0.207 ms which in comparison to the preliminary evaluation indicates that there is virtually no decline in performance as the index grows considering the number of global nodes is 1000 times greater. The performance for finding global nodes using different group sizes based on "e" values can be seen in FIGURE 31. This shows an initial large spike showing quick improvement due to caching. Another advantage of using globals over object ID key lookups is that the functions are already provided in Cache' object script to iterate over columns in the index.

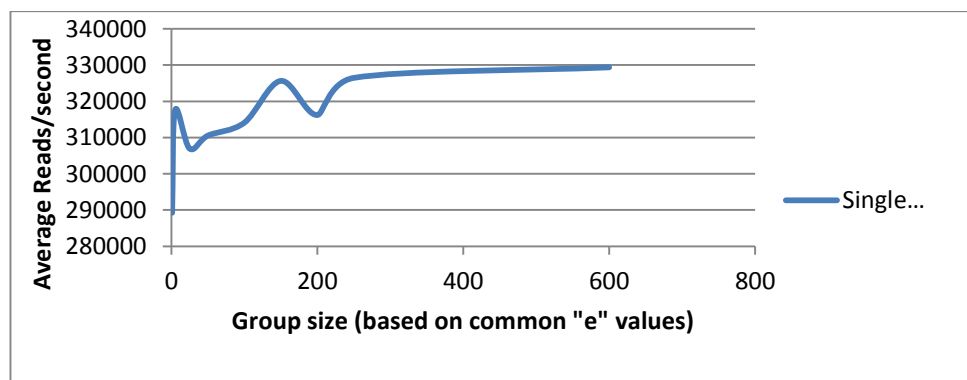


FIGURE 31 Average index read/second for globals within different group sizes

The results for a single global structure for storing the index information for increasingly larger databases certainly performed much better than expected. It was thought that perhaps other

techniques would be required such as using indirection in Caché to generate globals at runtime to reduce the scope of the search (As seen in FIGURE 33) . However after similar tests were ran with the code in FIGURE 32, it was found that the performance and storage space of this approach is much worse than a single global structure.

```

ClassMethod insert (start As %Integer, end As %Integer) {
    for e=start:1:end {
        for c=1:1:50 {
            for a=1:1:5 {
                for n=1:1:20 {
                    set cVar = "^e"_e_"c"_c
                    set @cVar@(a,n) = e_c_a_n
                }
            }
        }
    }
}

```

FIGURE 32 Code to store global structures using indirection

After inserting 116 million nodes for indexes, the size of the database using indirection was 18267 MB, Approximately 9 times larger than the equivalent amount of inserts on the single global structure. Also, the insert rate of nodes per millisecond for the single structure was almost 11 times quicker than the indirection approach. The performance of reads was also far from acceptable taking on average 109ms for the indirection database file containing only 23.2% of the number of global nodes.

STEP 1: Find a list of COMPOSITIONS, EHR_STATUS or EHR_ACCESS containing specified archetype

Global generated at design/compile time:

^ArchetypeLOOKUP (ehr_ID, container_uid, archetype_id) = ""

EHR IDENTIFIER	CONTAINER (COMPOSITION EHR_STATUS or EHR_ACCESS)	ARCHETYPE IDENTIFIER
<i>ehr id</i>	<i>uid</i>	<i>Globally qualified</i>
250c8300-d2 9b-41f4-a716- 378853450000	ab679eae-6074- 4871-8dda-561 8bbdbc210::7259 38E9-1C61-4C8 8-9BC62FC6 8731CFCB::1	<i>openEHR-</i> EHR- INSTRUCTION. referral.v1

STEP 2: Find a node corresponding to archetype details within an already known container ID

Global dynamically generated at compile time:

^ContainerXXXX (archetype_ID, archetype_node_id, archetype_number) = node_object_key

ARCHETYPE IDENTIFIER	ARCHETYPE NODE ID	ARCHETYPE CONTAINMENT
<i>Globally qualified</i>	<i>Unique within archetype</i>	<i>Unique within containment structure</i>
<i>openEHR-</i> EHR- INSTRUCTION	0 (root) at0002	0..*

FIGURE 33 Decomposing the search for archetype nodes in multiple steps using indirection

Given the positive results for the single global structure lookups it was decided to use this approach for use cases 4 to 6. Even though the lookup REF (id key) approach performed slightly better than globals, the flexibility gained from globals will be more beneficial. Insertion of the global nodes containing the path information can be achieved by storing a transient list of the path information to persist when versions are converted from OpenEhr.V1.Its.XML objects to the Caché proxy objects. The global structure was a slightly simplified version of the structures described earlier to account for the fact that only a single version of a composition is allowed in our prototype and testing. The structure can be seen in FIGURE 34.

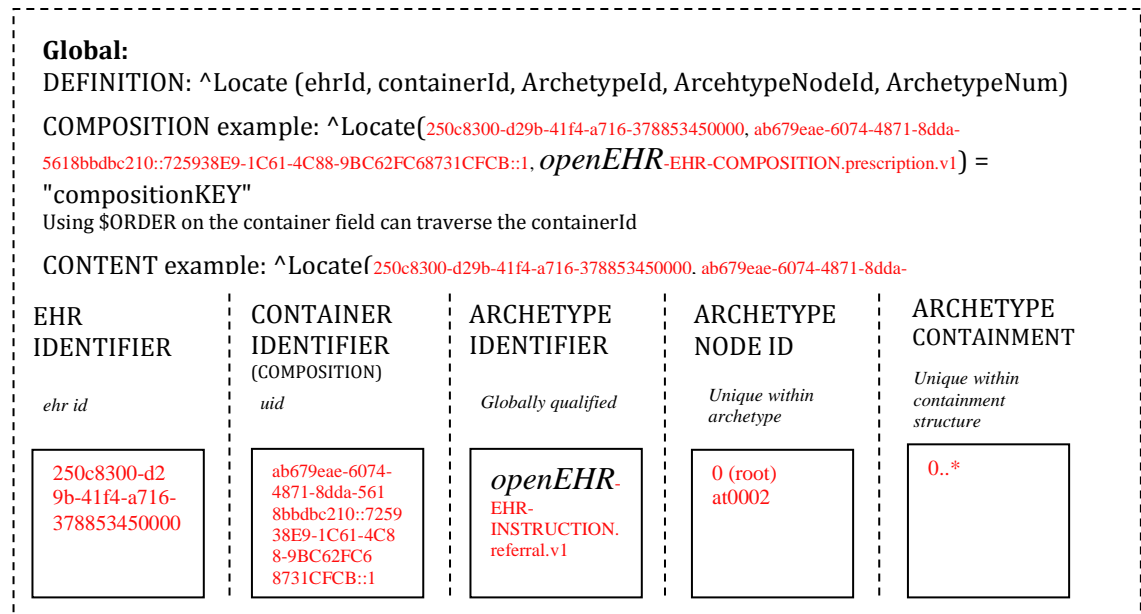


FIGURE 34 Global structure used to store path information of LOCATABLE objects in the prototype

The global structure in FIGURE 34 supports use cases 5 and 6 (described in *section 6.1.1*). Use case 4 can be achieved using a direct object ID lookup on the composition objects. The only use case this particular global structure does not support is use case 3 because it is for meta-queries about COMPOSITION objects. A similar structure can be setup for use case 3 by labelling all paths except for objects descendant from the "content" property in a COMPOSITION. Since there are relatively few and relatively shorter paths for the meta-data, they can be stored in a hash table on the client side in memory and mapped to compact integers for the global representation. However, since the performance of the global structures had already been addressed, Caché queries were used to satisfy use case 3. The declarative nature of queries, and ability to specify constraints in "WHERE" clauses, also make it the natural and easiest approach to use for queries on meta data.

6.1.3 Implementation Issues

Quite a few issues were identified using the .NET Managed Provider for Caché, which limited the ability to test it to the full capabilities. The "ListOfObjects" type in Caché could be stored and expanded quite easily. However, for some reason, despite importing the basic *openEHR* RM classes from a schema, there were problems retrieving Caché list objects from C#. The data in the array was empty and could not be Swizzled into memory. This issue was solved by creating a wrapper for the lists in Caché that provided enumerated access but only returned one object at a time.

Despite this setback, the wrapper proved useful in solving a second issue. Objects were returned in C# from Caché according to the type of the typed list. It was very difficult to cast some objects back from the base type to the sub type. In fact this problem was only partially solved and certain tests had to be omitted due to Target Invocation and Illegal CastExceptions that were left unsolved. Two approaches attempting to solve this, involved using the base class's name field to find the actual type of object. This information was used at compile time to hard code casts and at run time using dynamic method invocations. Other approaches such as delegates may solve the problem.

Regardless, we found that the .NET managed provider's extensive use of reflection and dynamic activation inhibits Caché's ability to perform to its potential. This can be illustrated by the contrast between the large transfer rates and scalability shown previously in comparison to the insert times and worse read performance for COMPOSITION objects reported in the final results. A tighter binding is needed to achieve better performance as clearly the few compositions that could be retrieved were often 6 times slower.

Implementation of the global nodes used to lookup values for LOCATABLE objects in *openEHR* was achieved by using two stacks. These structures keep track of the current hierarchy level as the conversion takes place between systems. Object ID's need to be generated throughout the traversal requiring additional save calls. However this is optimised when configured correctly.

6.2 Test Data

Test data from existing systems was unavailable. Instead, a set of 9 composition instances were generated and used for test data. The list of composition instances used are shown in TABLE 6

Corresponding Archetype	Size in XML	Serialised	Instances in each EHR
Temperature	7KB	7041 Bytes	10
Antenatal Referral	49KB	17897 Bytes	8
Data Set	1,319 KB	175647 Bytes	1
Current Problem	8KB	4959 Bytes	1
Blood Glucose	11KB	8626 Bytes	12
Blood Pressure	10KB	7846 Bytes	10
Pathology Report	30KB	14390 Bytes	6
Prescription	30KB	11547 Bytes	6
Discharge Summary	14KB	8865 Bytes	6
Total:	1,478 KB	250.7 KB	60

TABLE 6 The set of compositions residing in each EHR in the data set

The initial plan was to insert 2,400 EHR objects with 60 COMPOSITION instances each. Each EHR contains a mixture of the composition instances above to test an *openEHR* system on a larger end of the scale. These values have been arrived at based on statistics of AML data which predicts averages of 50 COMPOSITIONs per year at around 20KB each. 60 COMPOSITION objects was chosen due to the increasing rate of data capture.

To simplify the prototypes implementation, each VERSIONED_COMPOSITION object contained a single version of a COMPOSITION. In reality such rigorous testing was not at all possible due to the implementation issues found with the .NET Managed Provider. Originally, 7500 EHR instances were inserted into the MS SQL Fast Infosets implementation but due to the expectation that the same amount or equivalent tests could not be performed for Caché within a realistic timeframe

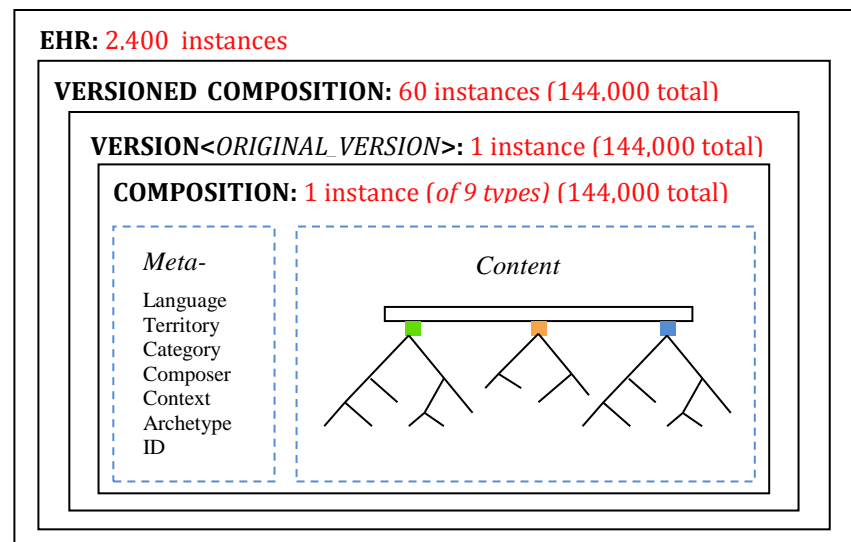


FIGURE 35 Visual description of the data to be stored in the database for testing

6.3 Test Environment

The server operating the databases is almost exactly the same as that used in the Preliminary Evaluation. The only difference is that a new empty hard-drive has been used to fulfil disk space requirements instead of the original used in the preliminary evaluation. The hard drive used is a Seagate Momentus 500GB with similar specifications, although slightly slower transfer rates, and access times. However the tests in the final results are compared independently of the preliminary

evaluation results. The disk access benchmarks of this hard drive measured with "HD Tune 2.55" can be seen in FIGURE 36.

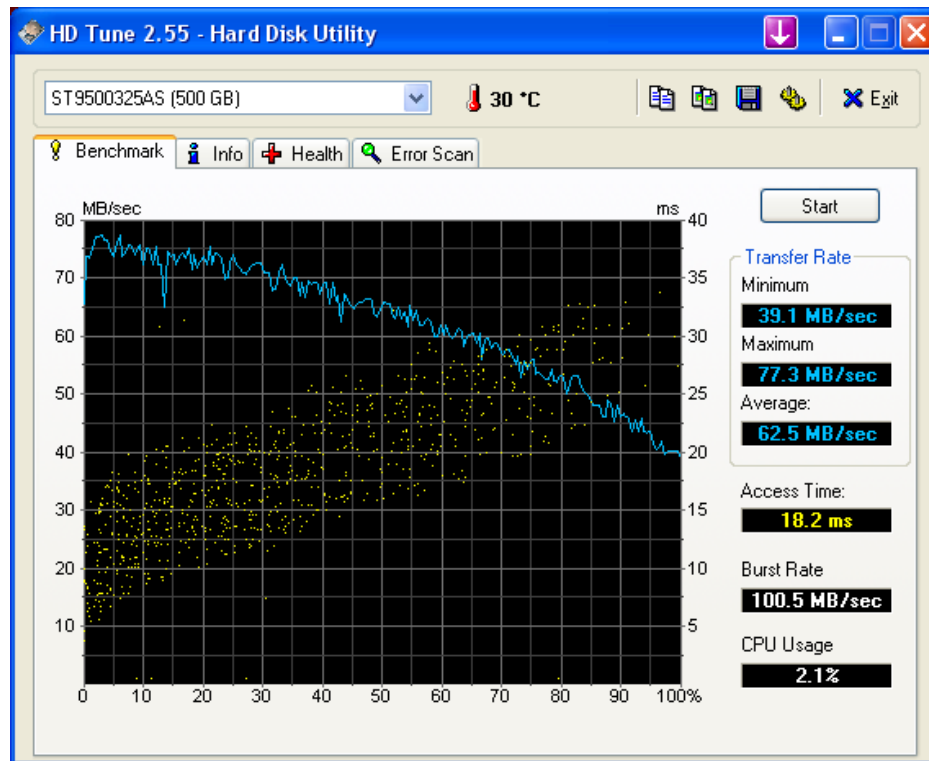


FIGURE 36 Results from 'HD Tune' Benchmark for the Seagate hard drive

It was initially considered to run the tests on multiple client machines with several threads to gain more throughput and test the database in a networked environment. However, a decision was made that only a single PC running both the server and client processes would be used in the testing. This is largely based around the decision to look at the performance and implementation advantages and disadvantages of OODBs used in *openEHR* in the practical evaluation sections of this paper.

More complex hardware and software architectures are required to test the potential scalability which is also dependant on the overall system architecture and requirements. This can be illustrated in papers looking at scalable architectures for scientific computing in systems such as GrayWulf (Szalay et al. 2009) and CERN (Holtman & Bunn 2000). Some of the systems evaluated in the aforementioned papers are quite different to healthcare however the domain is quite varied with very different requirements from use of *openEHR* in perhaps a GP office compared to bioinformatics. Earlier papers such as those by Martin Schaller (Schaller 1999) in the same CERN

project were able to gauge some performance and scalability indicators on minimal hardware architectures by analysing the trends found in the data collections.

6.4 Test Scenarios

It was originally planned to perform 5 tests repeatedly at a specific intervals starting with the insertion operations and followed by the query operations and tests. The reason for the intervals was to see how the performance was affected as the database became larger. It would also be useful to evaluate how the performance changes when the size of the database out grows the available memory allocated to the database. However, due to the implementation issues we faced in order to get comparable results, only one pass was performed on all databases with a smaller amount of EHR objects.

1. Insert 400 EHRs into the database with 60 compositions each measuring the time it takes to commit each individual composition. Each composition should be a separate transaction as this is the usual case in health systems. It is not likely that more than 3 or 4 will be inserted together.
2. Perform a query to retrieve the Meta information about composition in a particular patient's record for all HER objects. Repeat 6 times for a total of 2400 queries
3. Find a composition by UID on Blood Temperature compositions, Antenatal Referral and the Data Set which are all different sizes. Performing the tests on small and large objects allows us to see if the bottleneck is in finding the data or retrieving data from the physical disk and serialising it. Repeat 2 times for a total of 2400 test measurements.
4. Find a set of compositions for a patient based on the archetype specified. Perform on 3 different archetypes that correspond to different sized compositions. Repeat 2 times for a total of 2400 test measurements.
5. Find a set of content items for a patient within an archetype by its archetype node. This is a particularly important test for evaluating the alternative to path traversal for finding content.

Run the test with 3 different archetypes at nodes that are at the deepest hierarchical level in each different sized composition. Repeat 2 times for a total of 2400 test measurements.

The typical measurements of interest for performance include: write rate, query times for different scenarios, storage size and how all are affected by the size of the database.

6.5 Results and Comparison

6.5.1 Disk Space

FIGURE 37 shows a direct comparison of the database file size for each database tested after the insertion of 100 EHR objects. SQL Server's storage usage was the smallest which may be due to the compression used in Fast Infosets.

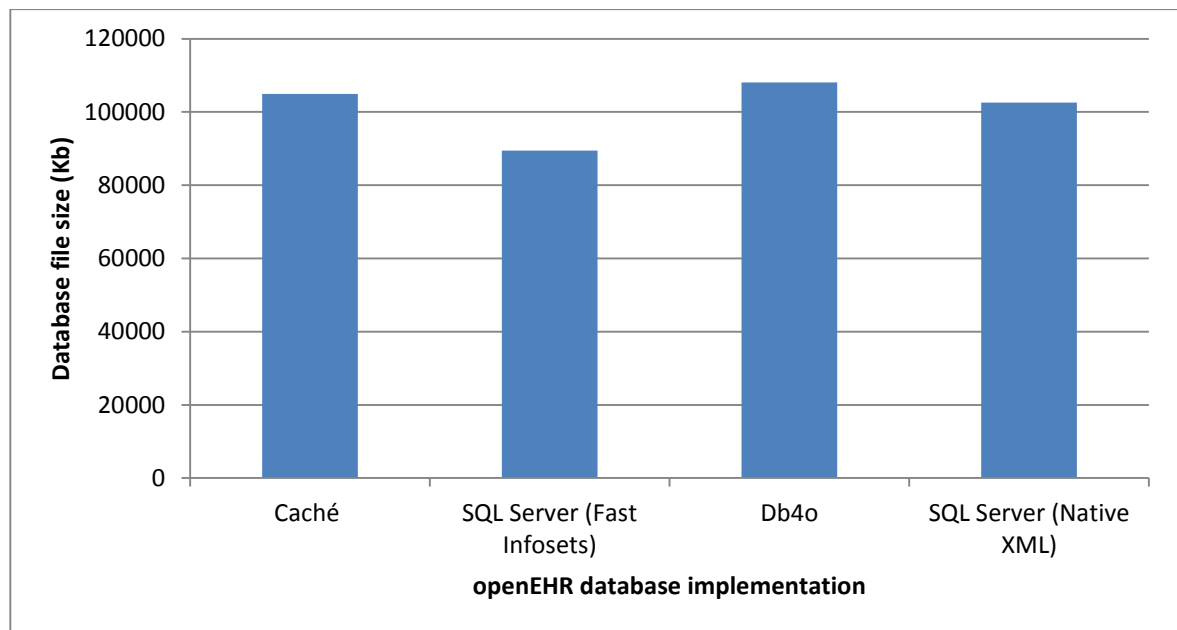


FIGURE 37 Database file size for 100 EHRs with 60 compositions

FIGURE 38 shows how the storage space of the MS SQL Fast Infosets increases in proportion to the number of instances being stored. This was to be expected, although it may not be so consistent when storing XML files as plain text.

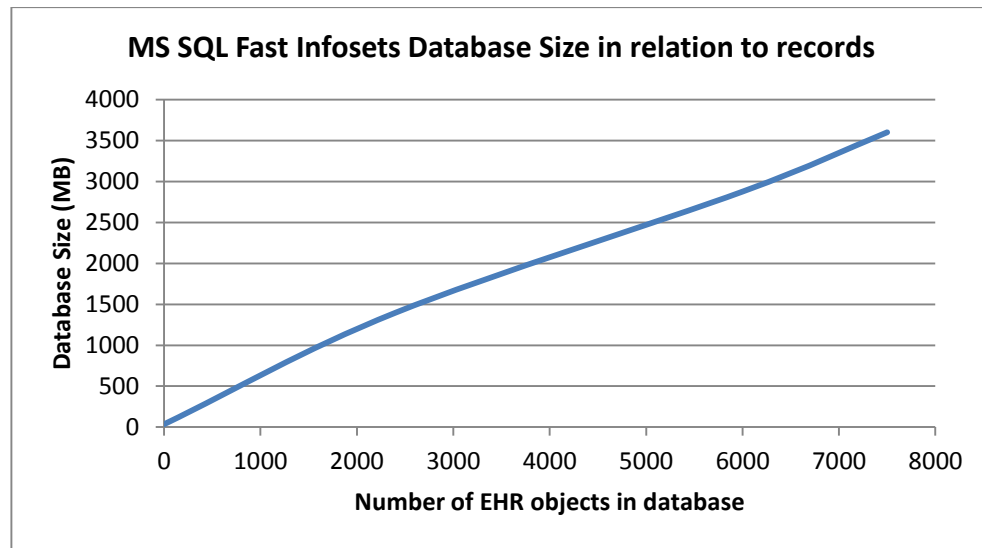


FIGURE 38 Storage space utilisation as the number of EHR objects grow in the database

6.5.2 Insertion

The results from the insertion tests showed large differences in the Object-Oriented database implementations to the Relational-XML Enabled databases. The Fast Infoset approach achieved slightly faster insertion times than the Native XML approach due to better compression algorithms (discussed in *section 3.2*).

The reason for the performance difference between Caché and MS SQL may be explained by observing *FIGURE 39*. The time to persist smaller COMPOSITION instances was actually quite similar to the relational databases. However, larger COMPOSITION instances such as the Discharge summary took over 10 times longer to commit to the database. The conversion layer for Caché requires complete object traversal to convert the objects and a significant amount of reflection is performed in the .NET managed provider. Furthermore, objects which exist lower in the content hierarchy are persisted separately, in multiple transactions, in order to obtain an object ID for the global structures described in *section 6.1.1*.

The idea of separating the indexes to the objects in Caché was supposed to improve access time and also insertion time. There may be ways to improve this in Caché by setting the object ID of each node as UUID and persist the objects in a single transaction.

Db4o, whilst performing better than Caché in some tests, did not compare well to the MS SQL at all. It was determined in SECTION 4.1 that db4o is more suitable for smaller scale embedded applications. It may not have the ability to handle the extent of the complex *openEHR* object model.

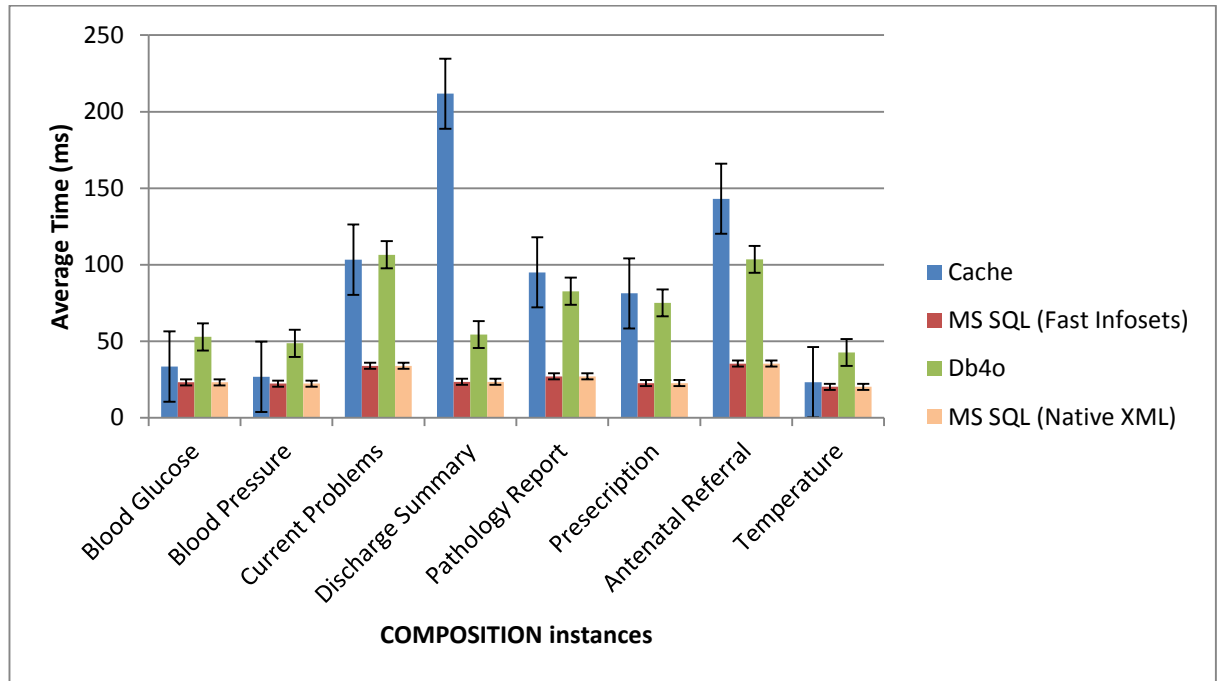


FIGURE 39 Comparison of the average time to persist single types of compositions in the first test pass (*with standard error*)

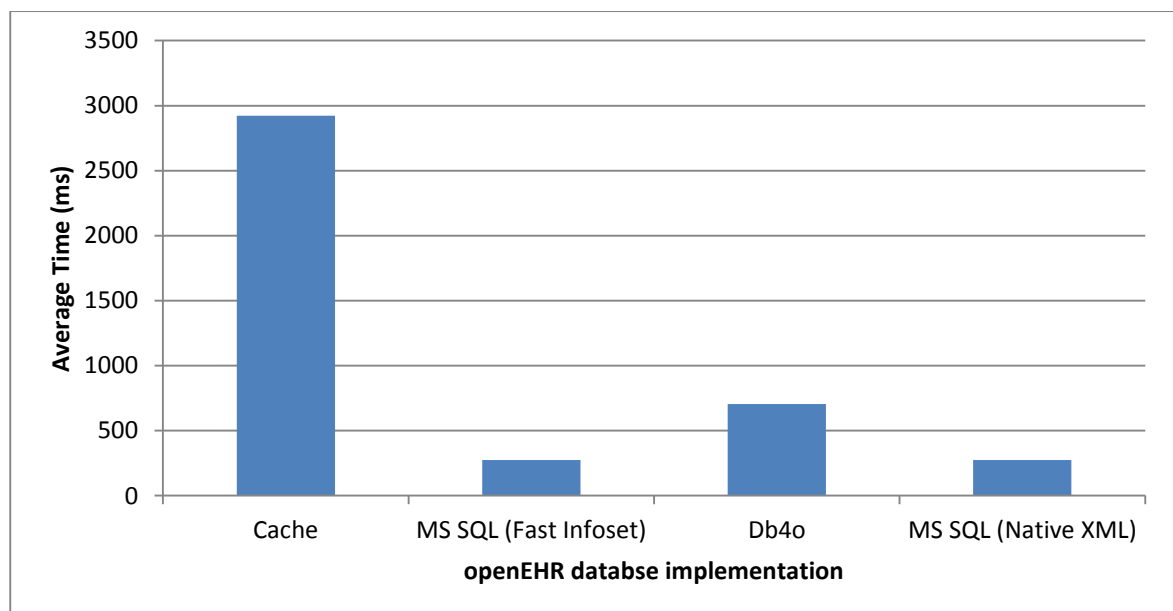


FIGURE 40 Comparison of the average time to persist a larger data set in to several *openEHR* implementations

6.5.3 Find and Retrieve a COMPOSITION's Meta-data

An increase in the amount of time required for querying meta-data in the Ocean Informatics solution occurred as the database size increased. Since there is the same amount of compositions in each EHR, this can only be caused by either the inability of indexes to perform as well in larger databases or as a result of the size of the database increasingly approaching the allocated memory size. However, given that the database was not over 3GB and there is a total of almost 8GB of RAM available in the system this is questionable. The initial configuration settings were used to provide a dynamic cache size depending on the size of the database and available memory. A subsequent test is needed on larger amounts of data or a smaller allocation of memory to confirm the cause of this behaviour. This test was not performed on Caché any other databases due to the problems encountered preventing the test from completed.

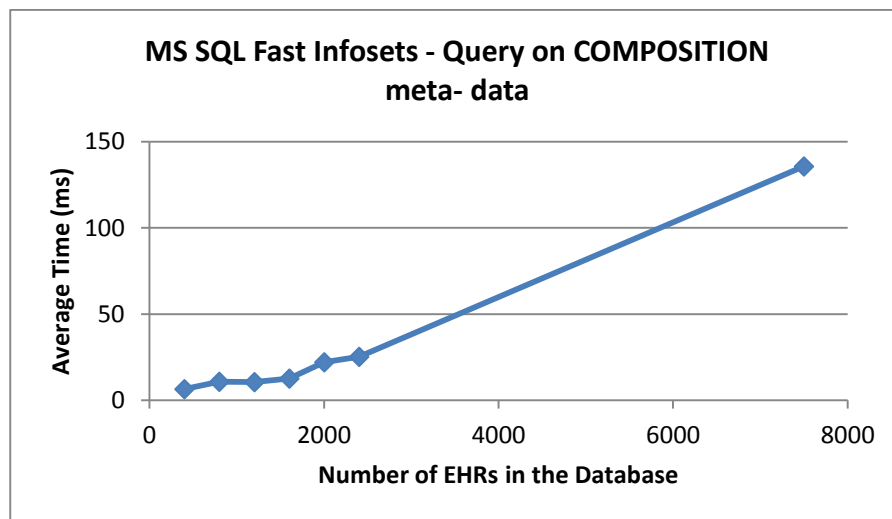


FIGURE 41 Performance of Microsoft SQL Server queries on Composition Meta-Data

6.5.4 Find and Retrieve a COMPOSITION by a unique identifier

In an object database these tests are basically a direct object lookup and the same operation in relational databases should be highly optimised due to indexing. If the index is built correctly and scalable most of the time required by this query is due to the physical disk access and the de-serialisation or conversion phases. This is illustrated in FIGURE 42. The values correspond to the serialised size of the objects where a referral is 10.18% the size of a data set and a temperature composition is 4.008% the size of a dataset. This also shows that performance and scalability of

lookup surrounded upon a single ID is just as feasible as the analysis of Caché global nodes in FIGURE 30.

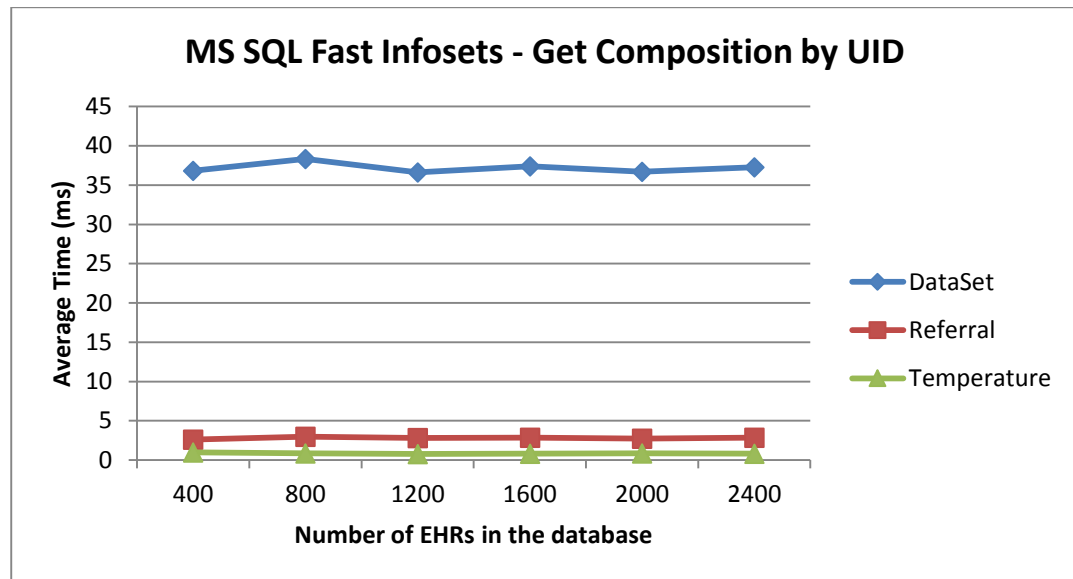


FIGURE 42 MS SQL (Fast Infosets): Avg. Time to retrieve a composition as the size of the database increases

FIGURE 43 compares the time needed to retrieve a composition by UID in all implementations, except for Caché. From the current tests we cannot explain why the time was the same for every type of composition in db4o. Again MS SQL (Fast Infosets) was clearly the fastest. The reason for the large decrease in performance in the MS SQL (Native XML) database for retrieving a dataset is due to increased amount of data to parse and de-serialize.

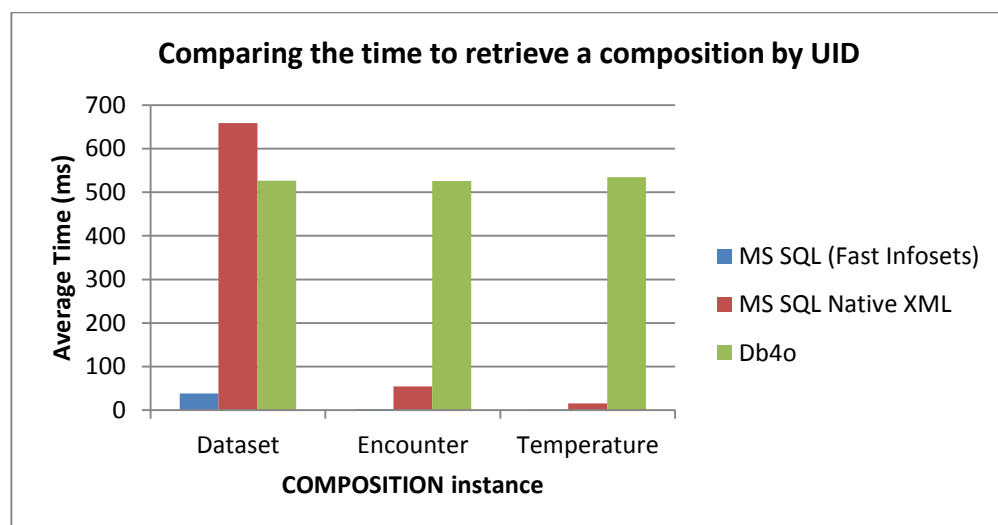


FIGURE 43 Comparing the avg. time to retrieve a composition by UID in db4o and MS SQL

6.5.5 Find and Retrieve a COMPOSITION based on its corresponding archetype

Due to the different number of types of compositions allocated in each EHR, it was expected that each sub test would yield similar results (results were averaged in SECTION 6.5.4). For instance there are 10 referrals but they are only 10.18% the size of a data set and only one data set was allocated to each EHR.

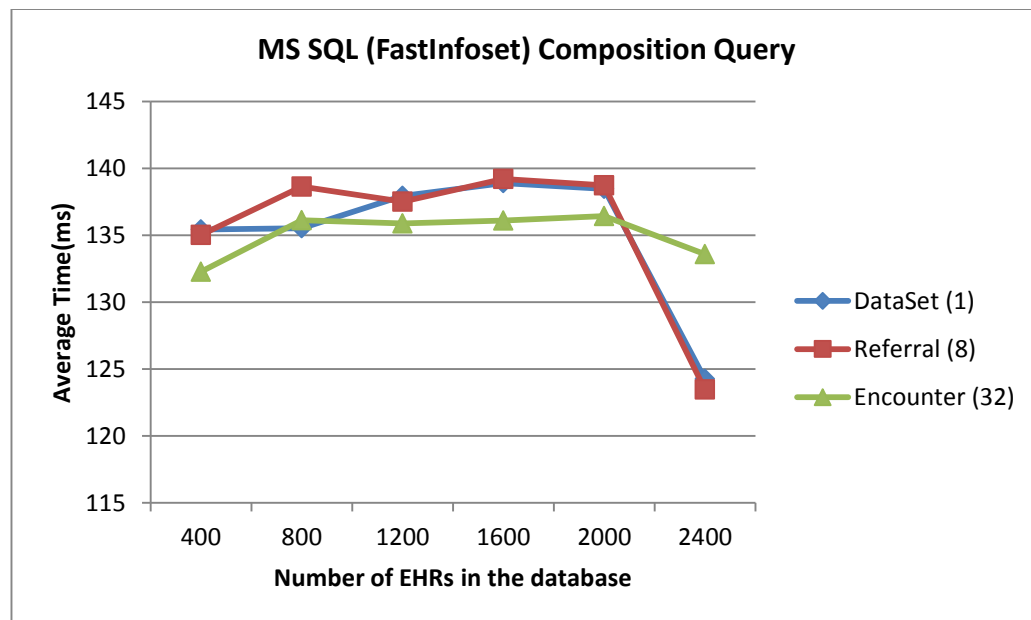


FIGURE 44 Performance of Microsoft SQL server for retrieving compositions based on archetypes

Reflection issues we found during would invalidate this test for Caché, so we decided not to attempt these measurements. The results comparing MS SQL and db4o are shown in FIGURE 45. The MS SQL Fast Infosets implementation was at least 5 times quicker than the db4o implementation in each test instance. Strangely, db4o performed better in this attempt than in the Composition by UID tests, where an index was placed on the “uid”. The activation level was checked and set to an order of magnitude much higher than the depth of the object trees to ensure the complete object was being returned.

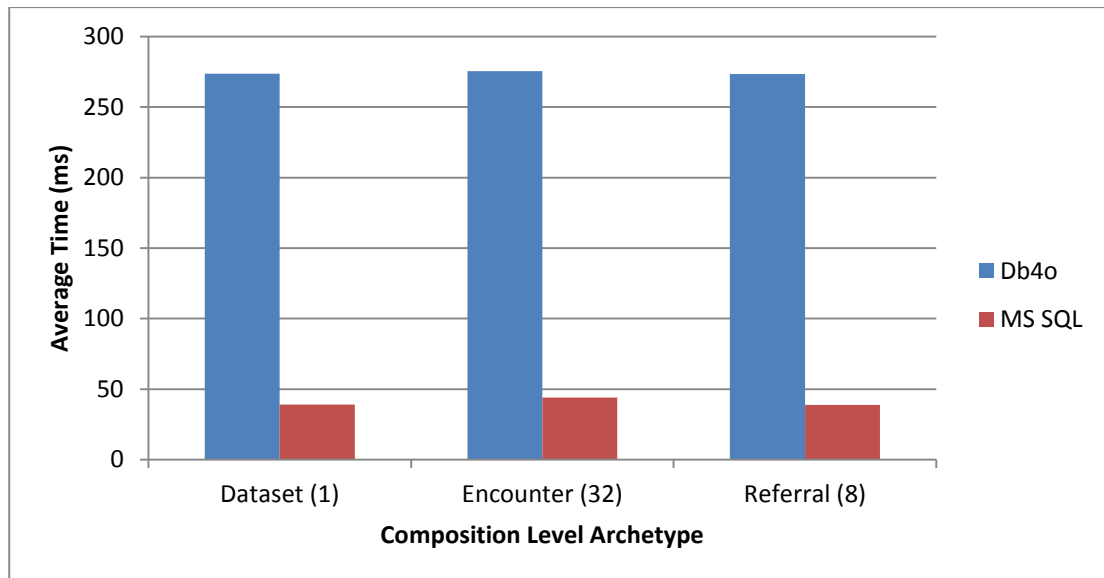


FIGURE 45 Comparative results for db4o and MS SQL on composition level queries

6.5.6 Find and Retrieve a CONTENT_ITEM based on the archetype and archetype node

Contrary to the initial criticisms of a traversal or parsing approach for finding content within an EHR, the tests performed on the Ocean Informatics implementation shows consistent results in the Fast Infosets implementation. The number of items returned in the result set of each content query depends on the number of archetypes of the specified type in one EHR. In order to simplify the comparison, the average time to retrieve a single content item from a single composition is shown.

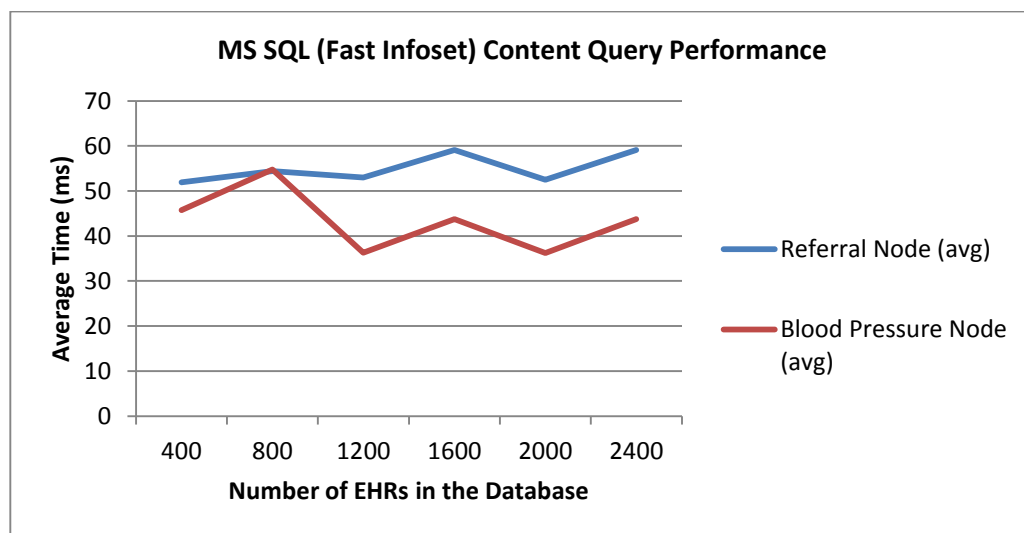


FIGURE 46 Performance of MS SQL (Fast Infoset): Content Queries in relation to the database size

Despite not being able to cast objects in the conversion classes created for the Caché persistence layer, the content queries could still be tested in another way. The archetype node in the data set to be returned is at id "at0025" within “*openEHR-EHR-OBSERVATION.substance_use.v1*” archetype (FIGURE 47). Since this is known, it is possible to manually activate the data (including meta data such as the node id) for testing purposes to obtain a measurement than with an entire composition.

```
<items xsi:type="ELEMENT" archetype_node_id="at0025">
  <name>
    <value>Interest in stopping</value>
  </name>
  <value xsi:type="DV_BOOLEAN">
    <value>true</value>
  </value>
</items>
```

FIGURE 47 Content at node id "at0004" within a specific archetype in the data set

As expected, the path labelling approach in SECTION 6.1 resulted in similar query time per test instance regardless of the depth that the content resides. FIGURE 48 shows the average time taken to retrieve a single node within an archetype. Both nodes were located at a depth of about 4 but the Referral Archetype is much larger and has an archetype composition hierarchy and a list of items for which the node is at the end. The approach used in Caché resulted in time measurements which were very similar for all content queries. However the MS SQL approaches require traversal which explains the increase in average query time for the more complex archetype. Particularly large data sets benefit from path labelling more than smaller datasets that require minimal traversal. The literature in SECTION 3 reported that Fast Infosets were faster at parsing XML than even the native XML storage which may explain the slower results for the MS SQL (Native XML) implementation.

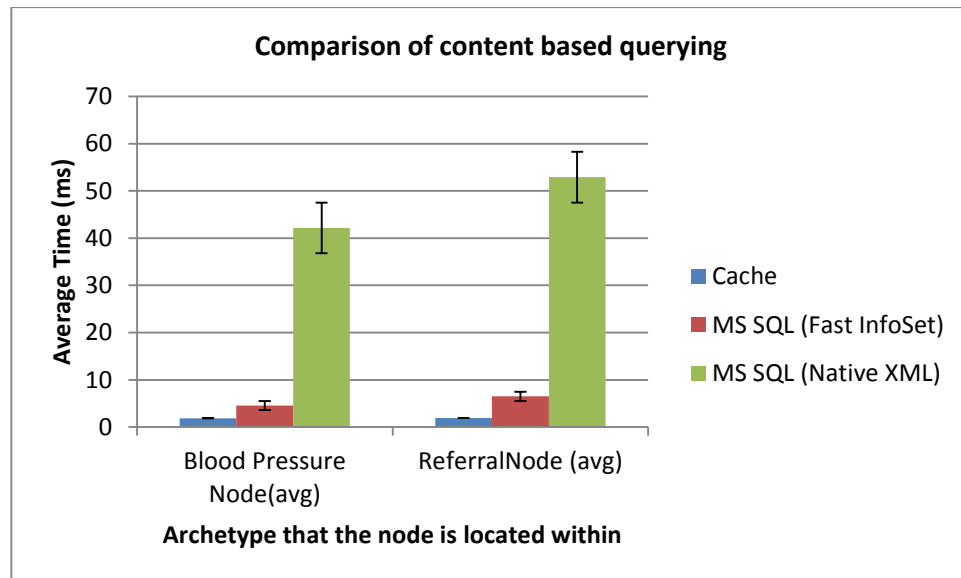


FIGURE 48 Comparison of the average time to retrieve a single node from an archetype (*with standard error*)

7 Discussion

Due to the implementation issues experienced with Intersystems Caché, the ability to do a complete analysis of results was limited. Complete test results for db4o and Intersystems Caché could not be obtained in the time available for the project. It has been quite clear throughout the project, that the results gathered upon db4o do not make it a good candidate for implementation as a persistence layer in *openEHR*. It is not really designed to handle the extensive object model and rigorous requirements of a large scale health care system.

Intersystems Caché, on the other hand, certainly has the features to be used as a basis for an EHR system. However, the issues identified with the C# binding do not make it a good candidate to use in an *openEHR* RM implementation. In an attempt to resolve the issues, a simpler object model was implemented and tested, which did actually work. For instance lists did in fact return values in smaller trial problems. The approach used to achieve a bridge between C# is certainly not optimal and it presents issues of a different kind to the object-relation impedance mismatch or XML serialisation and parsing issues. By contrast, the Microsoft SQL Server 2005 implementation still performed better in the tests where we were able to obtain results, with the exception of content queries.

There is no doubt that Caché has the potential to perform much better in the right context. For instance the test presented in FIGURE 30 achieved an average insertion rate of 12.2MB/second. Retrieval required much more time to find the object but once the object ID or reference to the object is found, read time appears to be limited, only by disk access and load due to concurrent connections. The difference in this test was that C# was only used to call a method on the cache server which handled the lookup and insertion.

The conversion from the OpenEhr.V1.Its.XML package to Caché requires a complete object traversal to copy the proxy object. It also requires a complete traversal of runtime assemblies when performing operations on the proxy objects. This is not much of an issue for those tests. However, the few results obtained on queries that actually allowed us to return a composition, always took over 500ms which is very slow in comparison to the results found for MS SQL Server. It would be highly recommended that if Caché is further considered for use in *openEHR*, the Jalapeño bindings

are evaluated due to its tight integration with the language. Ideally, the whole *openEHR* specification can be implemented in Caché and is a common approach in EHR systems (particularly with previous systems using MUMPS a predecessor to Caché). However there is little support for 3rd party tools, version control of source code and other standard software engineering tools to easily be used in team-based development. The effort required to manage these complexities with Caché may prove to be more difficult than handling the MS SQL Server mapping layers.

Personal communication with *openEHR* developers indicated that often a source of concern is the insert time and storage space more so than the query times due to highly optimised indexes. The results have shown that Fast Infosets used in MS SQL server provide much better storage efficiency than even very light weight, low overhead databases such as db4o. However this study only sampled a small portion of Object-Oriented databases and others such as Objectivity/DB which was discussed but not experimented with may provide better results.

The approach presented for labelling path nodes to provide quicker access than traversal was one area where the Caché implementation outperformed the MS SQL implementations. This same approach could be implemented in a relational database with all the path information stored in columns with a foreign key to the record storing the XML blob. As such it is certainly not only limited to Object-Oriented databases. The main difference is that, the ID in certain object databases provides a more direct approach to the data by reference. Although this largely depends on how the database engine is implemented.

8 Conclusion

Prior to the commencement of this study, literature regarding the suitability and potential for the use of the Object-Oriented databases in the *openEHR* Reference Model was rather scarce. The relational data model is not semantically rich enough to express the models provided in the *openEHR* specification in an efficient manner. For this reason, software projects implementing the *openEHR* specification turned to storing XML and later binary encoded XML as Fast Infosets into blobs on mature relational database products.

This study has investigated the potential use of object-oriented databases for the persistence layer of an implementation of *openEHR* systems as an alternative to current approaches such as the XML-Relational approach. One of the original problems is the time needed for parsing and serialisation of XML files, even when encoded as binary. The other issue of previous implementations of *openEHR* involves the development cost and maintenance of a custom XML based persistence layer,

After exploring three Object-Oriented database products, it has been found that those may present a different set of issues at the intermediate layer in comparison to parsing and XML serialisation. For instance bindings such as the .NET managed provider for Caché has a negative influence on the ability to use all the standard Object-Oriented features. Db4o is more useful in lightweight embedded situations as opposed to handling the *openEHR* Reference Model.

Object ID keys and global structures in Caché can improve performance of content based querying in *openEHR*. After investigating various implementation approaches, it was found that using path based information as indexes to archetype nodes, provides consistent and fast performance. These approaches should be scalable. If there are issues due to memory limitations, the separation of indexes from the data allows for optimisations such as selective indexing, top level version indexing and scope based indexing.

The performance of the *openEHR* RM prototypes implemented in Intersystems Caché and Db4o was compared to the implementation in MS SQL used by Ocean Informatics. The results showed that the MS SQL Server (Fast Infosets) implementation performed better on the majority of tests including insertion, size of database file and querying compositions and meta-data. However, the

path labelling approach implemented in Caché provided faster and more consistent performance than any other implementation on the content query tests.

These findings suggest that there is still potential for more research into object-oriented databases for *openEHR*. Use of Caché with another language binding or products such as Objectivity/DB should be considered which is proven to be scalable, but still offers tight integration with programming languages. The implementation techniques regarding the labelling of paths for direct access rather than traversal have a large potential to be useful in *openEHR* since the system. Since the specification involves a strict versioning system update or deletion of nodes in object hierarchies do not affect the path labelling.

9 References

Atkinson, M, Bancilhon, F, DeWitt, D, Dittrich, K, Maier, D & Zdonik, S 1989, 'The Object-Oriented Database Systems Manifesto'.

Austin, T 2004, 'The Development and Comparative Evaluation of Middleware and Database Architectures for the Implementation of an Electronic Healthcare Record', CHIME, University College London.

Bauer, MG, Ramsak, F & Bayer, R 2003, 'Multidimensional Mapping and Indexing of XML', paper presented at the German Database conference.

Beale, T 2008, *Archetype Query Language (AQL) (Ocean)*, OpenEHR, Adelaide, viewed September 25 2009, <[<http://www.openehr.org/wiki/display/spec/Archetype+Query+Language+\(AQL\)+\(Ocean\)>](http://www.openehr.org/wiki/display/spec/Archetype+Query+Language+(AQL)+(Ocean))>.

Beale, T 2002, *Archetypes: Constraint-based Domain Models for Future-proof Information Systems*.

Beale, T & Heard, S 2007a, *Archetype Definition Language*.

Beale, T & Heard, S 2007b, *Archetype Definitions and Principles*, openEHR Foundation.

Beale, T & Heard, S 2007c, *Architecture Overview*, openEHR Foundation.

Beale, T & Heard, S 2007, *An Ontology-based Model of Clinical Information*, Australia.

Beale, T & Heard, S 2007d, *The Template Object Model (TOM)*.

Beale, T, Heard, S, Ingram, D, Karla, D & Lloyd, D 2006, *Introducing openEHR*, openEHR Foundation.

Beale, T, Heard, S, Karla, D & Lloyd, D 2007, *EHR Information Model*, openEHR Foundation.

Becla, J & Wang, DL 2005, *Lessons Learned from Managing a Petabyte*.

Begoyan, A 2007, *An overview of interoperability standards for electronic health records*.

Bird, L, Goodchild, A & Heard, S 2002, *Importing Clinical Data into Electronic health Records - Lessons Learnt from the First Australian GEHR Trials*.

Bird, L, Goodchild, A & Tun, Z 2003, 'Experiences with a Two-Level Modelling Approach to Electronic Health Records', *Research and Practice in Information Technology*, vol. 35, no. 2, pp. 121-138.

Biryukov, A & Khovratovich, D 2009, *Related-key Cryptanalysis of the Full AES-192 and AES-256*, University of Luxembourg, Luxembourg.

Celko, J 2004, *Trees and Hierarchies in SQL for Smarties*, Morgan Kaufmann,

CEN 2008, *CEN: The European Committee for Standardization*,
<<http://www.CEN.eu/cenorm/homepage.htm>>.

Chaudhri, AB 1993, 'Object Database management Systems: An Overview'.

Codd, EF 1979, 'Extending the database relational model to capture more meaning', *ACM Trans. Database Syst.*, vol. 4, no. 4, pp. 397-434.

Codd, EF 1970, 'A relational model of data for large shared data banks', *Commun. ACM*, vol. 13, no. 6, pp. 377-387.

Connolly, T & Begg, C 2005, *Database systems: a practical approach to design, implementation, and management*, Addison-Wesley,

Conrick, M 2006, *Health Informatics: Transforming Healthcare with Technology*, Nelson Thornes Ltd

Cook, WR & Rosenberger, C 2006, 'Native Queries for Persistent Objects A Design White Paper', *Dr. Dobbs's*, 01 February 2006.

EFDSsoftware 2008, *HD Tune*, viewed 25 September 2009, <<http://www.hdtune.com/>>.

Florescu, D & Kossmann, D 1999, *A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relation Database*, pp. 31 - 38.

Gamma, E, Helm, R, Johnson, R & Vlissides, J 1994, *Design Patterns: Elements of Reusable Object-Oriented Software* Addison-Wesley Professional,

GridwiseTech 2007, *Implementing a High Performance Service Oriented Architecture: Integrating SOA with Objectivity/DB*, GridwiseTech, Krakow.

HL7 2008, *HL7 Healthcare Interoperability Standards*, <<http://www.hl7.org.au/>>.

Holtman, K & Bunn, J 2000, *Scalability to Hundreds of Clients in HEP Object Databases*, Addison Wesley Longman, Inc, pp. 11--2.

Hutchinson, A, Kaiserswerth, M, Moser, M & Schade, A 1996, *Electronic data interchange for health care*, pp. 28-34.

Intersystems 2009a, *Caché Data Integrity Guide*, Intersystems Corporation, Cambridge.

Intersystems 2009b, *Caché High Availability Guide*, Cambridge.

Intersystems 2009c, *Caché Security Administration Guide*, Intersystems Corporation, Cambridge.

Intersystems 2009d, *Caché System Administration Guide*, Cambridge.

Intersystems 2007, *InterSystems Caché® Technology Guide*, Intersystems Corporation, Cambridge.

Intersystems 2008, *InterSystems Online Documentation*, viewed 5 November 2008, <<http://docs.intersystems.com/cache20081/csp/docbook/DocBook.UI.Page.cls>>.

Intersystems 2009e, *Using the Caché Managed Provider for .NET*, Cambridge.

Kaiserslautern, TU 2009, *The XTC Project: Native XML Data Management*, Postfach, <<http://www.lgis.informatik.uni-kl.de/cms/dbis/projects/xtc/>>.

Khan, L & Rao, Y 2001, *A Performance Evaluation of Storing XML Data in Relational Database Management Systems*, Atlanta, pp. 31-33.

Larman, C 2005, *Applying UML and Patterns: An introduction to Object-Oriented Analysis and Design and Iterative Development*, Pearson Education,

Leslie, H 2007, 'International developments in openEHR archetypes and templates', *Health Information Management Journal*, vol. 37, no. 1, p. 2.

Leslie, H & Heard, S 2006, *Archetypes 101*, p. 6.

Lu, J 2009, 'Related-key rectangle attack on 36 rounds of the XTEA block cipher', *International Journal of Information Security*, vol. 8, no. 1, February 2009.

Ma, C, Frankel, H, Beale, T & Heard, S 2007, 'EHR Query Language (EQL) - A Query Language for Archetype-Based Health Records', *MEDINFO*.

Maldonado, JA, Moner, D, Tomas, D, Angulo, C, Robles, M & Fernandez, JT 2007, 'Framework for Clinical Data Standardization Based on Archetypes', *MEDINFO*.

Markl, V, Ramsak, F & Bayer, R 1999, *Improving OLAP Performance by Multidimensional Hierarchical Clustering*, IEEE Computer Society.

Microsoft 2009a, *File Systems*, viewed 1 September 2009, <[http://technet.microsoft.com/en-us/library/cc766145\(WS.10\).aspx](http://technet.microsoft.com/en-us/library/cc766145(WS.10).aspx)>.

Microsoft 2009b, *MySpace Uses SQL Server Service Broker to Protect Integrity of 1 Petabyte of Data*, Seattle, viewed October 15 2009, <http://www.microsoft.com/casestudies/Case_Study_Detail.aspx?casestudyid=4000004532>.

Microsoft 2007, *XML Best Practices for Microsoft SQL Server 2005*, <<http://msdn.microsoft.com/en-us/library/ms345115.aspx>>.

Microsoft 2009c, *XML Schema Definition Tool (Xsd.exe)*, MSDN, Seattle, viewed October 15 2009, <[http://msdn.microsoft.com/en-us/library/x6c1kb0s\(VS.71\).aspx](http://msdn.microsoft.com/en-us/library/x6c1kb0s(VS.71).aspx)>.

MIT 2009, *Kerberos: The Network Authentication Protocol*, Cambridge, viewed 26 August 2009, <<http://web.mit.edu/Kerberos/>>.

NEHTA 2007, *Standards for E-Health Interoperability*.

Nicola, M & John, J 2003, 'XML Parsing: A Threat to Database Performance', paper presented at the CIKM, New Orleans, Louisiana, USA.

Noemax 2009, *Fast Infoset Performance Benchmarks*, Noemax Technologies, Palaio Faliro, viewed October 15 2009, <http://www.noemax.com/products/fastinfoset/performance_benchmarks.html>.

Objectivity & Violin 2008, *A High Throughput Computing Benchmark of The Objectivity/DB Object Database and the Violin 1010 Memory Appliance*, Sunnyvale.

Objectivity, I 2005, *Hitting the Relational Wall*.

Objectivity, I 2006a, *Objectivity for Java Programmer's Guide Release 9.3*, Objectivity, Sunnyvale.

Objectivity, I 2009, *Objectivity Web-based Training*, viewed 1 September 2009, <<http://learn.objectivity.com/moodle/index.php>>.

Objectivity, I 2008, *Objectivity/.NET for C#*.

Objectivity, I 2006b, *Objectivity/DB Administration Release 9.3*, Objectivity, Sunnyvale.

Objectivity, I 2006c, *Objectivity/DB High Availability*, Objectivity, Sunnyvale.

Objectivity, I 2007, *Whitepaper: Objectivity/DB in Bioinformatics Applications*, Objectivity, California, p. 9.

Ocean Informatics 2008, viewed 1 October 2008, <<http://www.oceaninformatics.com/>>.

OceanInformatics 2006, *openEHR XML-schemas - Release 1.0.2*, openEHR, viewed 21 August 2009 2009,

Ohnemus, J 1996, *Evaluations of Object Oriented Databases for Storage and Retrieval of BaBar Conditions Information*, Stanford, Berkeley.

openEHR 2008a, openEHR, <<http://www.openehr.org/>>.

openEHR 2008b, openEHR SVN Archetype Repository, <<http://www.openehr.org/svn/knowledge/archetypes/dev/html/en/ArchetypeMap.html>>.

Paterson, J 2006, *The Definitive Guide to Db4o*, Science & Business Media, Berkeley.

Priti, M & Margaret, HE 1992, 'Join processing in relational databases', *ACM Comput. Surv.*, vol. 24, no. 1, pp. 63-113.

Rys, M 2005, *XML and relational database management systems: inside Microsoft SQL Server 2005*, ACM, Baltimore, Maryland.

Sandoz, P, Triglia, A & Pericas-Geertsen, S 2004, *Fast InfoSet*, Sun Microsystems, Santa Clara viewed October 15 2009, <<http://java.sun.com/developer/technicalArticles/xml/fastinfoset/>>.

Schaller, M 1999, *Objectivity/DB Benchmark*, CERN, Geneva.

Schloeffel, P, Beale, T, Hayworth, G, Heard, S & Leslie, H 2006, 'The relationship between CEN 13606, HL7, and openEHR', p. 4.

Shanmugasundaram, J, Tufte, K, He, G, Zhang, C, DeWitt, D & Naughton, J 1999, 'Relational Database for Querying XML Documents: Limitations and ', paper presented at the CLDB, Edinburgh, Scotland.

Shusman, D 2002, *Oscillating Between Objects and Relational: The Impedance Mismatch*.

Szalay, AS, Bell, G, Vandenberg, J, Wonders, A, Burns, R, Fay, D, Heasley, J, Hey, T, Nieto-SantiSteban, M, Thakar, A, Igen, Cv & Wilton, R 2009, *GrayWulf: Scalable Clustered Architecture for Data Intensive Computing*, IEEE, Waikoloa, Big Island, Hawaii.

Tian, F, DeWitt, DJ, Chen, J & Zhang, C 2002, 'The Design and Performance Evaluation of Alternative XML Storage Strategies', *SIGMOD Rec.*, vol. 31, no. 1.

Versant 2009, *dB4objects*, viewed 21 August 2009, <<http://www.db4o.com/>>.

WesternDigital 2008, *WD Caviar Blue*, Western Digital, Lake Forest.

Zloof, MM 1975, *Query-by-example: the invocation and definition of tables and forms*, ACM, Framingham, Massachusetts.

Zyl, PV, Kourie, DG & Boake, A 2006, *Comparing the performance of object databases and ORM tools*, South African Institute for Computer Scientists and Information Technologists, Somerset West, South Africa.

Appendix A: Performance Measurement Toolkit

The performance measurement toolkit was developed to help manage the logging and monitoring of database performance throughout the evaluation and testing. It is extendible so that other types of logging can be provided. However the use of the db4o logs allows retrieval of test results as objects which can then be de-serialised into whatever format is required such as .CSV or tab-delimited for further analysis. It is also configurable via. an XML file or method calls on the Config class. It was originally planned to include measurements such as the IO read/writes/other and other statistics from Windows Performance Counters but for the evaluation it was eventually determined that only the time elapsed measurements were required. The box below displays the typical code needed to setup a test and the following diagram is a simplified UML class diagram of the package. The ENUM and Exception classes have been omitted for conciseness. The idea behind this logger was to be extensible and allow easy transformation of data from the database. However a simpler CSV style streamlined approach was adopted for the final evaluation.

```
...
TestGroup testGroup = TestGroup.CreateTestGroup(DatabaseENUM.Cache, "Test
Find Blood Pressure", "Finds the systolic and dystolic blood pressure
measurements for a particular patient in previous encounters");

// Attach different types of loggers
testGroup.attachLogger(LoggerTYPE.UnformattedTxT);
testGroup.attachLogger(LoggerTYPE.Db4o);

// run multiple tests
for (int i = 0; i < 1000; i++)
{
    testGroup.startTest(MeasurementTYPE.INSERT, "Measurement Description",
0);
    ...
    testGroup.stopTest(123);
}

// the text logger appends throughout but this is required to save the db4o
// object currently
testGroup.saveAndFinish();
...
```

FIGURE 49 Code for setting up the logging facilities of a performance test

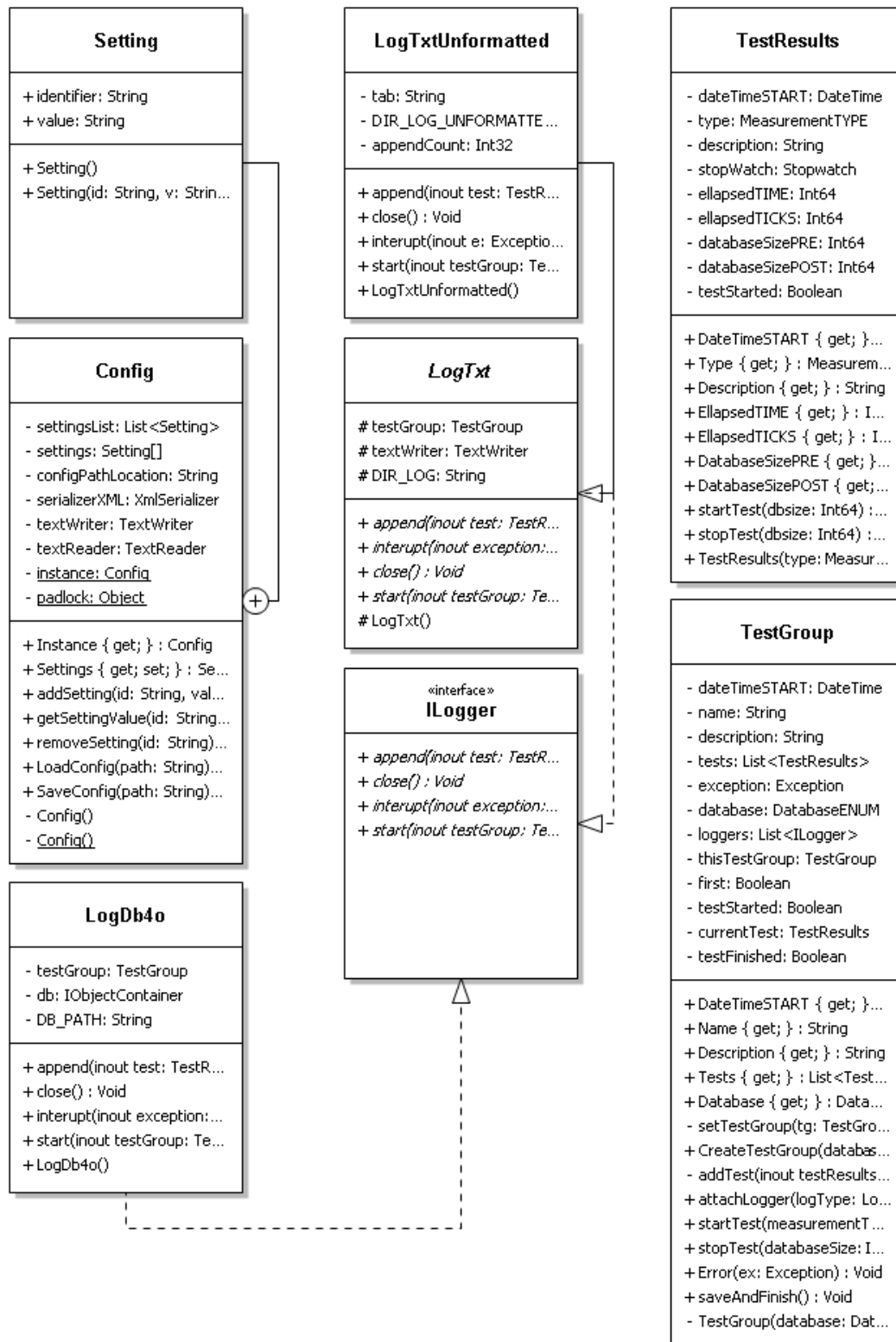


FIGURE 50 Simplified UML Diagram of Performance Monitoring Toolkit

Appendix B: Code used to manage globals and use cases

The code used to manage the globals in Caché is provided below. Population queries have been omitted as they are similar to the content queries but with an extra level of nesting. It would also be advisable to use a batch approach or enumerators for population queries on real data sets that may be very large.

```

Class openEhrV1.Get Extends %Persistent
{
    //
    // Find a composition within a specified EHR record by specifying meta-data as paths.
    // Retrieve the meta data as a result set
    //
    ClassMethod CompositionMetaDataSetSQL(ehrId As %String,
    compositionPaths As %Library.ListOfObjects, archetypeId As %String) As %Library.ResultSet
    {
        set result = ##class(%Library.ResultSet).%New()

        // process path lists
        set pathString = compositionPaths.GetAt(1)

        for i=2:1:compositionPaths.Count()
        {
            set as = i-1
            set pathString = pathString_"_"_compositionPaths.GetAt(i)

        }

        // Fetch a result
        do result.Prepare(sqlString)
        do result.Execute(ehrId)

        quit result
    }

    //
    // Find and retrieve a composition by a unique identifier
    //
    ClassMethod Composition(uid As %String) As openEhrV1.Composition
    {
        try
        {
            set objectKEY = ^locateCOMPOSITION(uid)
            set composition = ##class(openEhrV1.Composition).%OpenId(objectKEY)
        }
        catch exception
        {
            set composition = ""
        }
        quit composition
    }

    //
    // Find a composition by it's archetypes. Use global traversal to find the correct nodes
    //
    ClassMethod CompositionsByArchetype(ehrId As %String, archetypeId As %String)
    As openEhrV1.CustomList
    {
        set result = ##class(openEhrV1.CustomList).%New()

        set keyC = $ORDER(^locateLOCATABLE(ehrId,""))
    }

```

```

while keyC != ""
{
    try {
        set o = ^locateLOCATABLE(ehrId,keyC,archetypeId)

        if o != ""
        {
            // add result to a list to return
            do result.Insert(o)
        }

    } catch ex {
        // do nothing
    }

    // Go to the next composition
    set o = ""
    set keyC = $ORDER(^locateLOCATABLE(ehrId,keyC))
}

quit result
}

ClassMethod ContentByArchetype(ehrId As %String, archetypeId As %String, archetypeNode As %String)
As openEhrV1.CustomList
{
    set result = ##class(openEhrV1.CustomList).%New()

    set keyC = $ORDER(^locateLOCATABLE(ehrId,""))

    while keyC != ""
    {
        try {
            set keyA =
            $ORDER(^locateLOCATABLE(ehrId,keyC,archetypeId,archetypeNode,""))

            while keyA != ""
            {
                try {

                    // see if object exists
                    set o = ^
                    locateLOCATABLE(ehrId,keyC,archetypeId,archetypeNode, keyA)

                    if o != ""
                    {

                        // When a result is found add it to the result
                        do result.Insert(o)
                    }

                } catch ex {
                    // do nothing - will return null if there is no objects
                }

                set keyA =
                $ORDER(^locateLOCATABLE(ehrId,keyC,archetypeId,archetypeNode, keyA))
            }

        } catch ex {

        }

        // Go to the next composition
        set keyC = $ORDER(^locateLOCATABLE(ehrId,keyC))
    }

    quit result
}

```

Appendix C: Issues with Caché .NET Managed Provider

Some of these issues may in fact be due to issues with the implementation. There is quite a large amount of documentation available for administration, object scripting, globals and other Caché features. However, comprehensive documentation on the .NET managed provider was not available to assist in solving these problems. FIGURE 51 presents the first issue encountered which was lists returning null. A simplified object model of *openEHR* was tried which did in fact work and the source of this error could not be found.

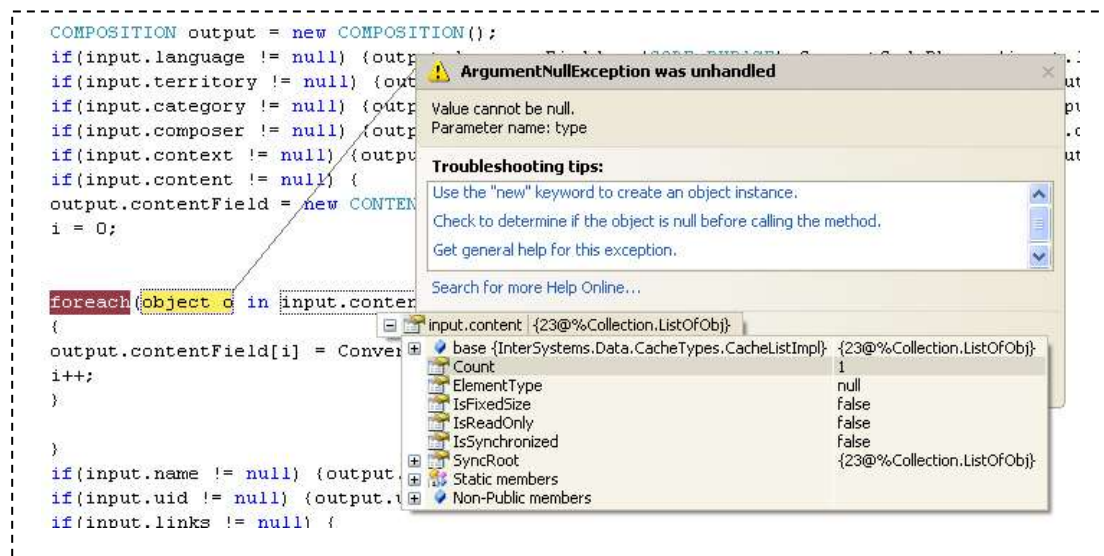


FIGURE 51 .NET managed provider: Lists that have items which contain no objects

One workaround which partially solved the problem involved creating a wrapper for the list which enabled the ability to get the object id of the items in the list open them. However since the type is only known from the class name it is stored as, further work is required using reflection to invoke the right methods to perform conversion from one object model to another. After trying to understand the proxy objects and the .NET managed provider code it may have been better to use:

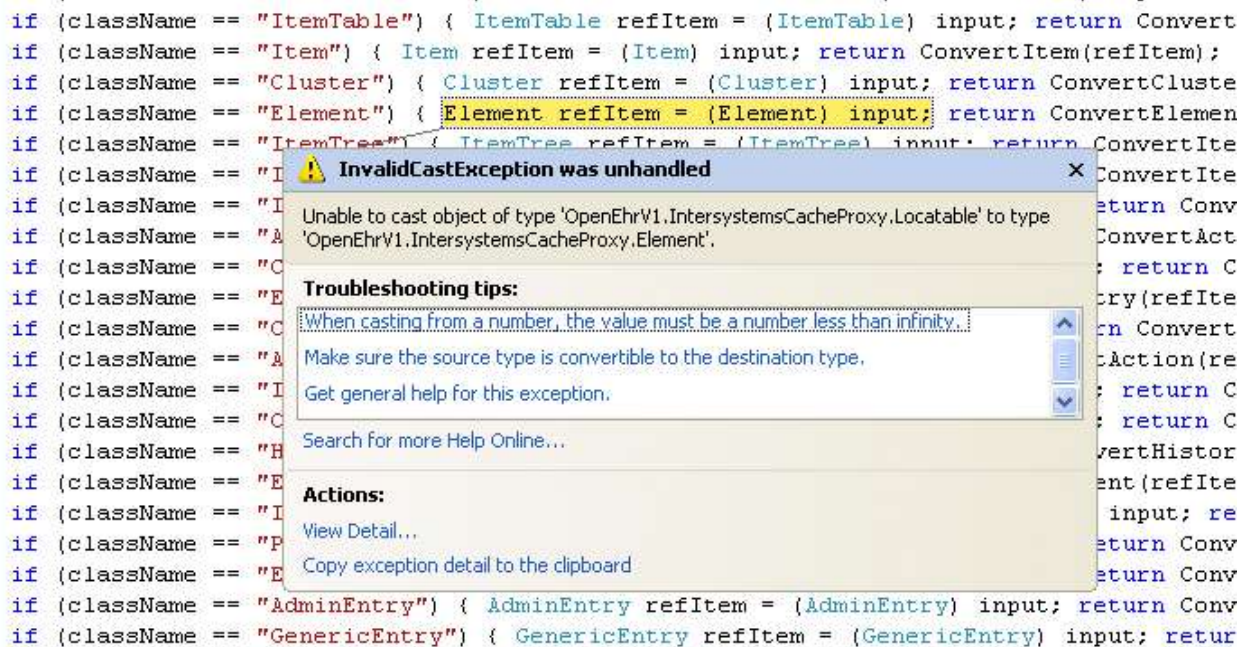
```
conn.OpenProxyObj(classname, id, typeof(type));
```

It seems that the facilities are there to cast types back to base objects, but it is difficult to manage with a conversion layer. This is unusual and unexpected behaviour is displayed in FIGURE 53, an alternative attempt to generate cast operations at compile time from the Code Generator program

instead of reflection and method overloading. On the other hand, db4o handles this seamlessly as shown in FIGURE 54. However, db4o is more suitable for embedded applications.

```
// Initialise array
OpenEhr.V1.Its.Xml.RM.CLUSTER[] cSharpObj = new OpenEhr.V1.Its.Xml.RM.CLUSTER[cacheObj.Count];
OpenEhrV1.IntersystemsCacheProxy.ListWrapper listWrapper =
    new OpenEhrV1.IntersystemsCacheProxy.ListWrapper(this.conn);
listWrapper.SetList(cacheObj);
string id;
int i = 0;
MethodInfo openMethod;
MethodInfo castMethod;
Type type;
while ((id = listWrapper.NextId()) != null) {
    type = convertCacheType(listWrapper.GetCurrentObjectName());
    openMethod = type.GetMethod("OpenId", new Type[] { this.conn.GetType(),
        typeof(string) });
    InterSystems.Data.CacheTypes.CacheObject o =
        (InterSystems.Data.CacheTypes.CacheObject) openMethod.Invoke(null,
            new object[] { this.conn, id });
    cSharpObj[i] = (OpenEhr.V1.Its.Xml.RM.CLUSTER) ConvertRM(o);
}
```

FIGURE 52 .NET Managed provider: One solution to the list problem, using a wrapper



```
if (className == "ItemTable") { ItemTable refItem = (ItemTable) input; return Convert
if (className == "Item") { Item refItem = (Item) input; return ConvertItem(refItem);
if (className == "Cluster") { Cluster refItem = (Cluster) input; return ConvertCluste
if (className == "Element") { Element refItem = (Element) input; return ConvertElemen
if (className == "ItemTree") { ItemTree refItem = (ItemTree) input; return ConvertIte
if (className == "I
if (className == "I
if (className == "A
if (className == "C
if (className == "E
if (className == "C
if (className == "A
if (className == "I
if (className == "C
if (className == "H
if (className == "E
if (className == "I
if (className == "P
if (className == "E
if (className == "AdminEntry") { AdminEntry refItem = (AdminEntry) input; return Conv
if (className == "GenericEntry") { GenericEntry refItem = (GenericEntry) input; retur
```

InvalidCastException was unhandled

Unable to cast object of type 'OpenEhrV1.IntersystemsCacheProxy.Locatable' to type 'OpenEhrV1.IntersystemsCacheProxy.Element'.

Troubleshooting tips:

- When casting from a number, the value must be a number less than infinity...
- Make sure the source type is convertible to the destination type.
- Get general help for this exception.
- Search for more Help Online...

Actions:

- View Detail...
- Copy exception detail to the clipboard

FIGURE 53 Showing the Invalid Cast operation which the .NET Managed Provider threw.

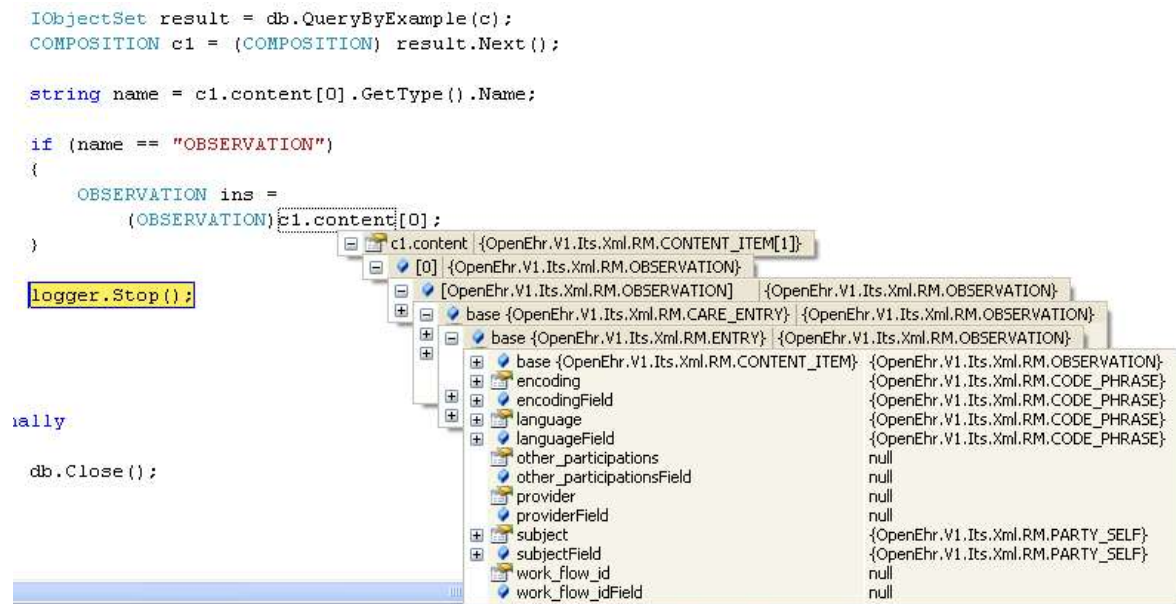


FIGURE 54 Db4o providing the ability to cast objects back to their original sub types

Appendix D: Code fragments from the code generator

The following is some of the example code from one version of the code generator that was created to convert types from the .NET managed provider proxy classes to OpenEhr.V1.Its.XML.

Code managing the content item labelling:

```
namespace OpenEhrV1.IntersystemsCacheProxy
{
    public partial class RMConverter
    {
        public string ehrID = "";
        public string containerID;
        public Stack<string> currentArchetype = new Stack<string>();
        public Stack<int> currentArchetypeNum = new Stack<int>();
        private List<string> archetypesVistited = new List<string>();

        public void reset()
        {
            ehrID = "";
            containerID = "";
            currentArchetype = new Stack<string>();
            currentArchetypeNum = new Stack<int>();
            archetypesVistited = new List<string>();
        }

        public int getArchetypeCount()
        {
            return currentArchetypeNum.Peek();
        }

        public string getArchetypeRoot()
        {
            return currentArchetype.Peek();
        }

        /// <summary>
        /// Put at start of the convert methods which are sub types of locatable
        /// </summary>
        /// <param name="node">The node at this level</param>
        /// <returns>The name of the archetype root for this node</returns>
        private void PreCheckArchetype(string node)
        {
            // just a normal archetype node, return top of stack
            if (node.Length > 2) {
                if (node.Substring(0, 2) == "at")
                {
                    return;
                }
            }

            // new archetype node, add to stack and list
            currentArchetype.Push(node);

            int count = 0;
            foreach (string s in archetypesVistited)
            {
                if (s == currentArchetype.Peek())
                    count++;
            }
            currentArchetypeNum.Push(count);

            archetypesVistited.Add(node);
        }

        /// <summary>
        /// Put at the end of the convert methods which are sub types of locatable
        /// </summary>
        /// <param name="node">The archetype node at this position</param>
    }
}
```

```

private void PostCheckArchetype(string node)
{
    // just a normal archetype node, leave it
    if (node.Length > 2)
    {
        if (node.Substring(0, 2) == "at")
        {
            return;
        }
    }

    // archetype node is root, remove it
    currentArchetype.Pop();
    currentArchetypeNum.Pop();
}
}
}

```

Code for generating methods to convert from proxy and net objects:

```

public string GenerateNETtoPROXY(Type t)
{
    if (t.IsEnum)
        return "";

    Type newT = assemblyNET.GetType(t.FullName);
    StringBuilder s = new StringBuilder();

    s.AppendLine("public " + PROXYType(t) + " Convert"+t.Name+"(ref " + t.FullName + " input)");
    s.AppendLine("{");

    s.AppendLine("string className = input.GetType().Name;");

    // FIRST CHECK SUBTYPES
    foreach (Type tInAssembly in RMtypes)
    {
        bool subclassOf = tInAssembly.IsSubclassOf(newT);
        if (subclassOf)
        {
            s.AppendLine("if (className == \"" + tInAssembly.Name + "\") { " + tInAssembly.Name + " ");
            s.AppendLine("    refItem = (" + tInAssembly.Name + ") input; return Convert" + tInAssembly.Name + ");");
        }
    }

    if (!t.IsAbstract)
    {
        // ADD CODE FOR LOCATABLEs (needed for creating global nodes for locatable items
        // ALSO FOR COMPOSITIONs for the uid use cases

        s.AppendLine("int i = 0;");
        s.AppendLine("this.PROXYType(t) + " output = new " + this.PROXYType(t) + "(this.conn);");

        if (newT.Name == "COMPOSITION")
        {
            s.AppendLine("output.ehrId = ehrID;");
            s.AppendLine("this.containerID = input.uid.value;");
            s.AppendLine("PreCheckArchetype(input.archetype_node_id);");
        }
        else
        {
            if (newT.IsSubclassOf(typeof(LOCATABLE)))
            {
                s.AppendLine("PreCheckArchetype(input.archetype_node_id);");
            }
        }

        // NORMAL CODE

        FieldInfo[] fields = t.GetFields(BindingFlags.NonPublic | BindingFlags.Public | BindingFlags.Instance);
        foreach (FieldInfo f in fields)
        {
            string escape = "";

```



```

        string refA = "ref";

        // PRE-FIELD checks
        if (f.Name == "namespaceField")
            escape = "@";

        if (f.FieldType.Name == "long" || f.FieldType.Name == "long?" || f.FieldType.Name == "Int32" ||
            f.FieldType.Name == "int" || f.FieldType.Name == "int" || f.FieldType == typeof(Nullable)
            || f.FieldType.Name == "int64") {
            refA = "";
        }

        if (f.FieldType.Name == "dataField")
        {
        }
        else if (f.FieldType.Name.Substring(f.FieldType.Name.Length - 2, 2) == "[]")
        {
            s.AppendLine("if(input." + escape + f.Name + " != null) {" + codeForArrayNETtoPROXY(f) +
                "}")
        }
        else if ((f.FieldType.Name == "DV_PROPORTION"))
        {
            s.AppendLine("if(input." + escape + f.Name + " != null) {" + "output." +
                this.ConvertFieldToPROXY(f.Name) + " = ConvertLong(" + refA + " input." + f.Name + ");
                }");
        }
        else
        {
            s.AppendLine("if(input." + escape + f.Name + " != null) {" + "output." + escape +
                this.ConvertFieldToPROXY(f.Name) + " = Convert" + f.FieldType.Name + "(" + refA +
                " input." + escape + f.Name + "); }");
        }
    }

    // PERFORM THE POST-CHECKS on Compositions and Locatable items
    if (newT.Name == "COMPOSITION")
    {
        s.AppendLine("output.Save();");
        s.AppendLine(PROXY_RM + ".WriteGLOBALS.CompositionLevel(this.conn,this.ehrID, t
            his.containerID, this.getArchetypeRoot(),output.Id());");
        s.AppendLine("this.reset();");
    }
    else
        if (t.IsSubclassOf(typeof(OpenEhr.V1.Its.Xml.RM.LOCATABLE)))
        {
            s.AppendLine("input.objectKEY = Guid.NewGuid().ToString();");
            s.AppendLine(PROXY_RM + ".WriteGLOBALS.LocatableLevel(this.conn,this.ehrID, t
                his.containerID, this.getArchetypeRoot(),
                input.archetype_node_id,this.getArchetypeCount().ToString(),output.Id());");
            s.AppendLine("this.PostCheckArchetype(input.archetype_node_id);");
        }
        s.AppendLine("return output;");
    }

    s.AppendLine("return null;");
    s.AppendLine("}");

    return s.ToString();
}

```