UNIVERSITY OF DUBLIN
TRINITY COLLEGE

DISSERTATION

PRESENTED TO THE
UNIVERSITY OF DUBLIN, TRINITY COLLEGE
IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MAGISTER IN ARTE INGENIARIA

Content Interoperability and Open Source Content Management Systems

An implementation of the CMIS standard as a PHP library

Author:
MATTHEW NICHOLSON

Supervisor:
DAVE LEWIS

Submitted to the University of Dublin, Trinity College,
May, 2015

Declaration

I, Matthew Nicholson, declare that the following dissertation, except where otherwise stated, is entirely my own work; that it has not previously been submitted as an exercise for a degree, either in Trinity College Dublin, or in any other University; and that the library may lend or copy it or any part thereof on request.

May 21, 2015

Matthew Nicholson

Summary

This paper covers the design, implementation and evaluation of PHP library to aid in use of the Content Management Interoperability Services (CMIS) standard. The standard attempts to provide a language independent and platform specific mechanisms to better allow content management systems (CM systems) work together.

There is currently no PHP implementation of CMIS server framework available, at least not widely.

The name given to the library is Elaphus CMIS.

The implementation should remove a barrier for making PHP CM systems CMIS compliant. Aswell testing how language independent the standard is and look into the features of PHP programming language.

The technologies that are the focus of this report are:

CMIS A standard that attempts to structure the data within a wide range of CM systems and provide standardised API for interacting with this data.

PHP A general-purpose scripting language that is typically used for web development.

Zend Engine Zend Engine the commonly used interpreter for PHP. This is responsible for running PHP code.

HHVM A virtual machine designed for running PHP and Hack (another programming language similar to PHP) code. HHVM has been developed by facebook as an alternative to the Zend Engine.

Wordpress A CM system, or website manager, that started out as blogging software. Wordpress will server as the platform the Elaphus CMIS will be used on for testing.

This paper cover some of the design decision that went into creating Elaphus CMIS. Aswell as details from the benchmarks designed to evaluate Elaphus CMIS.

Using the Elaphus CMIS in Wordpress provided largely positive results. The benchmarks performed showed that the generation of Atom CMIS documents is at least comparable to the generation of Wordpress web pages. In some cases the CMIS documents used less CPU time to generate.

The achieved request rates were better in the case of CMIS documents, meaning that as well as causing less of a load the CPU they also could be generated faster.

Acknowledgements

I would like to thank my supervisor Dave Lewis for his help and support through the project.

I must thank my family for their support and understanding during this past year, I deeply appreciate them.

Finally, I'd like to thanks my classmates and friends. They have been there to make this year enjoyable. And to thank Conor Brady who has been a good friend since my first year of college.

Abstract

The nice thing about standards is that you have so many to choose from.

– Andrew S. Tanenbaum, 1981

This paper covers the design, implementation and evaluation of PHP library to aid in use of the Content Management Interoperability Services (CMIS) standard. The paper examines language independence of the CMIS standard by applying to a PHP content management system and details the benchmarks performed to evaluate the server.

Parts of the CMIS standard where implemented in PHP and used within a Wordpress site. In particular basic navigation of post, pages and media through the Atom binding of the CMIS standard.

Strides have been made toward an implementation of a PHP library to allow PHP CM system be CMIS compliant. Benchmark show that the cost of the generations of CMIS objects in a PHP CM system requires less computation (CPU load) and time than the generation of a full HTML web page.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The title “An implementation of the CMIS standard as a PHP library” raises two immediate questions:

- What is CMIS?
- Why PHP?

This chapter aims to answer these questions and provide insight into the motivation for this dissertation. Further it covers a brief description of: what a CM systems is, what the CMIS standard is, and the choice of PHP for the implementation.

1.1 Background

A core concept of this dissertation is the concept of a content management system (CMS or CM system). This is a system for publishing or maintaining content that is stored in content repository or database. As Kampffmeyer [2006, p. 2] notes:

> The general term Content Management itself has a great many facets, and also includes Web Content Management, Content Syndication,

Digital or Media Asset Management, and naturally Enterprise Content Management as well. This "vicious circle" of terminology merely points up the lack of clarity in manufacturers' marketing language.

The two types of CM systems that the average user may come across are:

Web content management system (WCM) Allows for the management of a the content from a website. Websites that are managed by some form of WCM, included sites like Wikipedia, newspapers or typically any site with content written by people.

Enterprise content management (ECM) Kampffmeyer [2006] goes on describe the changing definition of what an ECM is and provides the application areas they cover, which includes: Document Management, Collaboration, WCM, Record Management and Workflow.

1.1.1 CMIS

Content Management Interoperability Services (CMIS) is a standard from the Organization for the Advancement of Structured Information Standards (OASIS). There have been two major versions. CMIS V1.0 which was approved on May 1 2010. The second and current version of is V1.1, which was first approved and published on 12 November 2012[1]. Details of this version will be provided later in this paper. As stated in the abstract "It (the CMIS standard) is intended to define a generic/universal set of capabilities provided by a CM system and a set of services for working with those capabilities" [TC, 2013, p. 2] Essentially the standard attempts to provide a universal structure and means to access the data within a CM system.

[1]https://www.oasis-open.org/news/announcements/content-management-interoperability-services-cmis-version-1-1-approved-and-publis

An important aspect of this type of standard is adoption, there is little benefit from having a standard that allows systems to work together if very few systems comply to it. Looking at the groups that make up the technical committee can give a sense high level of the interest in the problem of interoperability. There is a list of people and the organisation they are associated with is made available on OASIS's website[2] the organisations are state in table 1.1.

Accenture	Hewlett-Packard	Nuxeo
Adobe Systems	IBM	Oracle
AIIM	ISIS Papyrus America Inc.	RSD
Alfresco Software	Liferay, Inc.	SAP AG
ASG Software Solutions	Microsoft	Wells Fargo
EMC	Northeastern University	

Table 1.1: Members of CMIS technical committee

Müller et al. [2013] lists the following compliant systems:

- Alfresco
- EMC Documentum
- HP Autonomy Interwoven Worksite
- IBM Content Manager 8.4.3
- KnowledgeTree
- Magnolia CMS
- Microsoft SharePoint Server
- Nuxeo Platform
- OpenText ECM
- SAP NetWeaver Cloud Document Service

The standard provides a structure that can be used to represent the

[2]https://www.oasis-open.org/committees/membership.php?wg_abbrev=cmis

data within a CM system. This is expressed as the domain model which consisting of two parts

- Data Model, the representation of the entities within a repository.
- Services, a set of actions that can be taken to interact with these entities.

This structure is intent to be applicable to any content management system providing base level of functionality. The standard does not attempt to provide a full representation of all possible functionalities available in CM systems. It focuses on specifying a subset of functionality that is expected of a CM system. Some capabilities of the standard are marked as optional.

Content management systems have developed without standardisation. This has resulted in the situation that making content from one system available from another can require bespoke software acting as a connector between systems. This is due to systems being developed by different vendors. Problems can occur when systems are updated, if the structures of their data changes or even due to small changes to their APIs. Creating a standard API would make it easier for systems to work together. This is the problem of interoperability that the CMIS standard attempts to solve.

The standard defines three binding, means to expose the data from a CMIS server over the internet. CMIS V1.0 include Restful AtomPub and Web Services. Both of these are XML based technologies. CMIS V1.1 introduced the Browser Binding, a JSON based interface for CMIS servers.

Apache Chemistry

Apache Chemistry is a project from the Apache Software Foundation that provides an open source implementation of the standard. This includes

CMIS client libraries, a CMIS Server Framework and some development tools.[3]

As part of this client libraries are available in:

- Java
- Python
- PHP
- .NET
- Objective-C
- JavaScript

While the server framework is only available in Java. Apache Chemistry OpenCMIS is collection of providing the Java server and client.

1.1.2 PHP

PHP is an open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML[4]. PHP can be used for either procedural or object oriented programming, the details of these will be discussed in greater detail later in this paper.

PHP can be primarily consider a server side scripting language, it has been used to power different content management systems. Some common systems built on PHP are:

Drupal Originally a messaging board[5]. Drupal is released under the GPL licence. Drupal focus on providing a core functionality while being extendible through modules[6]. The current stable release is V7.36.

[3]http://chemistry.apache.org/
[4]http://php.net/manual/en/intro-whatis.php
[5]https://www.drupal.org/about
[6]https://www.drupal.org/mission

MediaWiki a content management system developed for Wikipedia and other Wikimedia Foundation projects. These are focused on user submitted content. [7]

TYPO3 is an open source enterprise CMS[8], as apposed to the others in this which are more focues on web content.

Wordpress originally was started as a "publishing system built on PHP and MySQL". Wordpress is a CMS with some focus on blogging but also works for more general purpose websites.

The implementation of this project will be applied to a Wordpress site. Wordpress is available from their website, wordpress.org. The software is available to that it can be used on the user's server.

The website wordpress.com is serivice that provides host of Wordpress web sites. They claim to provide an easy way to create a website.

This paper is concerned with the implementation of CMIS for web site using software from *wordpress.org*

1.2 Motivation

An implementation of the standard in PHP will remove a barrier for making PHP CM systems CMIS compliant. As mentioned, adoption of this type of standard is particularly important, making CMIS more available could aide in the success of the standard.

Implementing the standard will also serve to test how language independent it is. PHP is a dynamically typed language run in an interpreter. This offers a point of comparison to Java, which is strongly typed and is compiled and run within a virtual machine. //

[7]https://www.mediawiki.org/wiki/Project:About
[8]https://typo3.org/

CMIS structures data, this has implications beyond interoperability. Projects like DBpedia aim to extract structured data from Wikipedia. This results in a dataset that is semantically queryable. CMIS has a query mechanism that provides a relational model of the data within a content repository. CMIS could be used as a means to structure the data with in Wikipedia as well as external sources allowing more source be used in this Linked Data.

The standard does not limit itself to document management systems. Applying the standard to Wordpress which does not necessarily structure data in a hierarchical fashion offers another test of the possible applications of the standard.

OhAirt [2013] looks into the CMIS as mechanisms for locationation, finding that the CMIS standard could be extended to allow for localisation of content between CM systems,

1.3 Outline

Chapter 2 Provides information about CM systems, how CMIS as standard function. Also discusses PHP as programing language and some of the technologies built around it.

Chapter 3 Cover the design and implementation of CMIS in PHP, and the application of this implementation to a PHP CM system (Wordpress).

Chapter 4 Discusses the evaluation and benchmarking of the implementation mentioned in chapter 3.

Chapter 5 Discusses the future work in completing the implementation of CMIS for PHP as well as some of the potential research topics relevant to CMIS.

Chapter 2

State of the Art

This chapter details: what a CM system is, how the CMIS standard works and covers aspects of the Apache Chemistry Project. Details of PHP and some of the technologies related to it are also discussed.

2.1 What is a CM System

The CMIS standard attempts to provide a generalised structure for data within a content repository and a set of services to working with that data. As mention the definition of what a CM system is some what unclear. The standard attempts to provide an interface to a *repository*. This raises the issue of what the standard attempts to standardise. It does not attempt to cover all features of ECM repository.

Kampffmeyer [2006] discussed the state of ECMes, this is before the formations of committee for CMIS in late 2008[1]. The CMIS standard it does not restrict itself to a type of repository. The points Kampffmeyer makes about the interesting when consider what CMIS attempts.

[1] http://www.aiim.org/Research-and-Publications/Standards/Articles/CMIS

Kampffmeyer [2006] discusses the difference between a CMS and ECM, saying the claimed benefit of WCMes is broken into three points:

Integrative Middleware The concept of an Enterprise Application Integration (EAI) providing a single interface to multiple services, or maintaining consistency within different systems.

Independant Services For a given application one general service is available for that purpose.

Federated Repository an ECM is used as a content warehouse, storing information without regard for its source or use.

This raises the question of how CMIS can be used for these aims. How can CMIS work so that one client can retrieve data from multiple CMIS compliant sources? Would one CMIS compliant server be a solution, with other CM systems retrieving the information from that single source?

2.2 CMIS

The CMIS domain model is the means for structuring data within a CM system. This is done in two parts the data model and services.

2.2.1 Data Model

The aim of the data model is to provide a format that any CM system could use to structure its data. To do this CMIS represents the objects as fitting into one five base types:

Document These are used to describe information entities within a system.

Folder Allows for collections of fillable objects, providing a means to structure objects in a hierarchical manner.

Relationship Allow for a semantic dependency. that consist of a source and target object.

Policy represents an administrative policy that can be enforced by a repository. The CMIS standard does not provide an instances of policy objects, but a means for a repository to declare policies.

Item This was a new type introduced in CMIS v1.1, allows for objects which do not fit other object types. An given is sub types for an identity object could be declared for users and groups.

Additionally CMIS offers secondary object-types, which are a means to add extra data to an object. A CMIS repository MUST support the document and folder object-types.

These object types provides sets of properties, data fields, about that object. Properties may contain single or multiple values. There are properties that are common to all objects types, the base types then define additional properties for objects of that type.

Secondary object types allow for properties to be added or removed from an object. An object may only belong to one object type but can have many secondary object-types applied to it. Secondary object types allow for temporary data to be added to an object such as workflow, or can be used to add metadata to a document for example.

Figure 2.1 shows the UML diagram for these types. For example any CMIS Object has a cmis:objectId and cmis:objectTypeId. A cmis:document has the additional properties cmis:contentStreamFileName and cmis:contentStreamId. A document could represent an object that is binary data, such as an image. If a document has a binary stream these properties are used to identify that data. As a document may not require binary data, could represent plain text, these properties may be "not set" or contain an empty value.

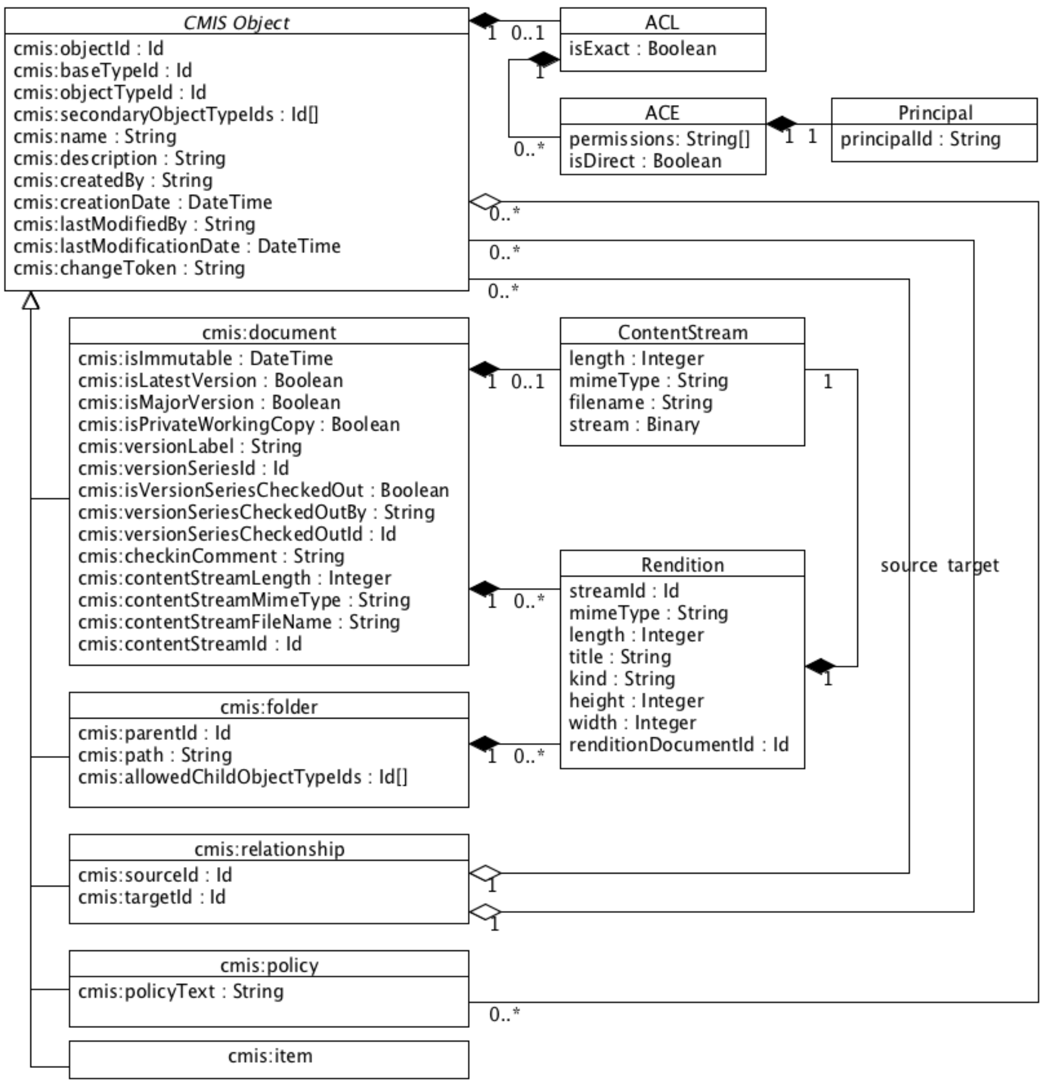
CMIS Object
cmis:objectId : Id
cmis:baseTypeId : Id
cmis:objectTypeId : Id
cmis:secondaryObjectTypeIds : Id[]
cmis:name : String
cmis:description : String
cmis:createdBy : String
cmis:creationDate : DateTime
cmis:lastModifiedBy : String
cmis:lastModificationDate : DateTime
cmis:changeToken : String
ACL
isExact : Boolean
ACE
permissions: String[]
isDirect : Boolean
Principal
principalId : String
cmis:document
cmis:isImmutable : DateTime
cmis:isLatestVersion : Boolean
cmis:isMajorVersion : Boolean
cmis:isPrivateWorkingCopy : Boolean
cmis:versionLabel : String
cmis:versionSeriesId : Id
cmis:isVersionSeriesCheckedOut : Boolean
cmis:versionSeriesCheckedOutBy : String
cmis:versionSeriesCheckedOutId : Id
cmis:checkinComment : String
cmis:contentStreamLength : Integer
cmis:contentStreamMimeType : String
cmis:contentStreamFileName : String
cmis:contentStreamId : Id
ContentStream
length : Integer
mimeType : String
filename : String
stream : Binary
Rendition
streamId : Id
mimeType : String
length : Integer
title : String
kind : String
height : Integer
width : Integer
renditionDocumentId : Id
cmis:folder
cmis:parentId : Id
cmis:path : String
cmis:allowedChildObjectTypeIds : Id[]
cmis:relationship
cmis:sourceId : Id
cmis:targetId : Id
cmis:policy
cmis:policyText : String
cmis:item
source
target
1
0..1
0..*

Figure 2.1: UML diagram for objects in CMIS.

Properties are typed and must fall into one the following types: string, boolean, decimal, integer, datetimes, uri, id, or html. Properties have attributes, pieces of metadata, associated with them such as: name, display name, default value, possible values and many more. Some attributes of a property are dependent on its type such as the precision of a decimal value. Figure 2.2 provides the UML for type definitions, this shows the attributes relevant to properties.

CMIS allows for the definition and mutability of object types. A system can define new object types, that must inherit of another object-type and have a base-type as an ancestor. This allows for object types with more properties to fit the need a specific system.

It is also possible to include additional properties, without declaring a sub-type. In the CMIS standard at the start of each object type has a section on property definitions. The opening sentence of each of these sections is to the effect of TC [2013, p. 34]:

> The document base object-type MUST have the following property definitions, and MAY include additional property definitions.

Similarly the server and client may ignore properties that they do not understand.

The CMIS standard defines some enumerated types, these are either returned by the services or taken as parameters to them. For example the getObject service has an optional parameter of type “enum IncludeRelationships”, the values of this type are declared in the standard.

2.2.2 Services

The services are the set of actions that can be taken on the data within a CMIS repository. They are the means the retrieve and manipulate the data within the repository.

They are grouped in the following way:

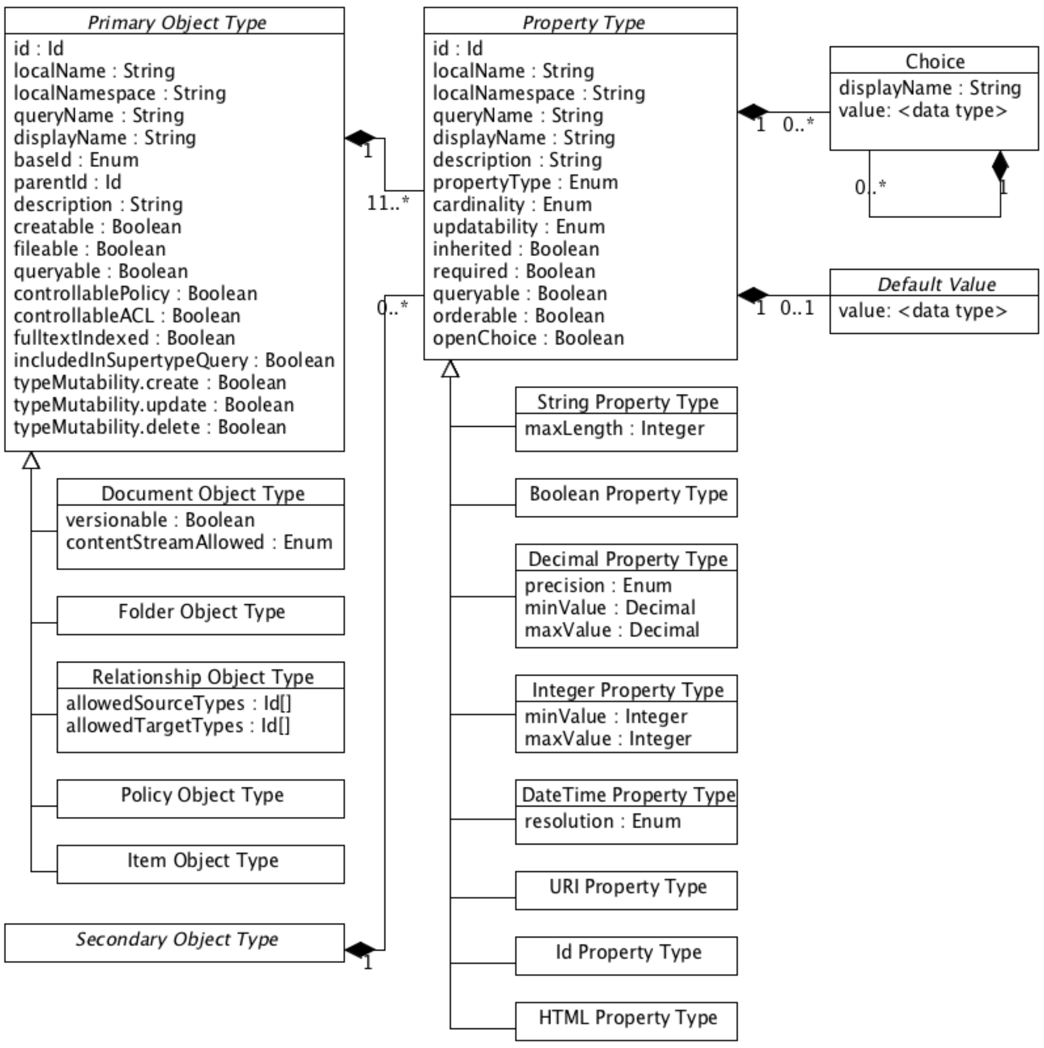

Figure 2.2: UML diagram for types in CMIS.

Repository Services Allow for discovery about the repository and the meta data about the types defined within the repository. Repositories may support type mutability, repository services also includes the methods of manipulating type definitions.

Navigation Services Allow for the discovery of the hierarchical structure of the data within the repository.

Object Services Allow for CRUD (Create, Retrieve, Update, Delete) operations on the objects within the repository.

Multi-filing Services Provides the ability to update objects when a repository supports multi-filing or unfiling optional capabilities

Discovery Services Expose the search functionality covered by the standard. CMIS provides an optional query capability that uses an SQL read-only projection of the CMIS data model into a relational view. This maps the properties of object-types to the attributes of a relational database structure.

Versioning Services CMIS cover the record of versions of objects within a repository as well as the concept of a private working copy.

Relationship Services Provides “getObjectRelationships” to give access to the relations associates with an object.

Policy Services Mechanism to apply, remove or get the policies associated with an object.

ACL Services Access Control Lists, CMIS provides a set of base set permissions as well as allowing for repository specific permissions to be applied to objects.

2.3 Apache Chemistry

As mentioned previously Apache Chemistry servers as the reference implementation of CMIS, the project provides a server framework and client libraries. OpenCMIS is a collection of Java libraries, frameworks and tools. The server framework aims to “hide” the detail of the bindings.

OpenCMIS has many parts, including an android client, general client libraries, a common library for clients and the servers, the server which includes a query parser and binding implementations, test utilities, a workbench for testing compliance, a sample server that stores data in memory. The packages of particular interest for this project are:

org.apache.chemistry.opencmis.commons

.data Interfaces for the objects in the data model, such as a container for the list of objects filed under a folder, as well as properties and objects.

.definitions contains the interfaces for the "definitions" within data model. This includes object-type and property-type definitions.

.enums This is the declarations of the Enums defined in the CMIS standard as well as some enums used internally within the OpenCMIS implementation. OpenCMIS also includes additional enums such as "DateTimeFormat" which is not mentioned in the standard.

.exceptions These are the exceptions as declared in the CMIS standard. CMIS standard declares two types of exceptions that can be raised from the services. General and specific exceptions, any service may raise a general expectation some services have specific exceptions they can throw.

.impl.dataobjects contains the implication of interfaces for the data model. The classes implement interfaces from both the ".data" and ".definitions" packages.

.spi This contains the CMIS service interfaces.

2.4 PHP

PHP is a scripting language commonly used in web applications. The description for the PHP packaging in Arch Linux is "An HTML-embedded scripting language".[2] It is a language with a focus on web development. PHP is dynamically typed, variables do not require declaration and PHP does not provide static type checking.

Though its history PHP has gone through some rewrites. PHP 3 was the first version to resemble the current versions of PHP. PHP 4 introduced the Zend engine, the two stage process for parsing and executing code. PHP 5 was released in 2004 and introduced a new OOP mode and a second version of the Zend engine. PHP is currently maintaining support for version 5.4, 5.5 and 5.6.

PHP is made up of two separate pieces, Golemon [2005] provides an introduction to the life cycle of a request being handled by PHP powered by the Zend engine. The Zend Engine is responsible for interpreting the script and producing machine code. It handles memory management and function handling and such. The PHP Core is responsible for communication with the Server Application Program Interface (SAPI). It talks to the application server, which may be a Apache http server or fastCGI.

Code for PHP may be written in two ways:

PHP script These are interpreted which can lead to inefficiency, the opcode from these scripts can be caches to reduce this cost, by avoiding the need to parse and compile the code every time the script is called. "Zend Opcache" is built into PHP since version 5.5, though it may not be enabled by default.

Extensions These have a use case of getting code to run smaller, and consume less memory while running.

Zend Opcache was added on the 20th June 2013[3].

[2]https://www.archlinux.org/packages/extra/i686/php/
[3]http://php.net/ChangeLog-5.php

Extensions can be written for PHP in C. These will appears as functions that are callable from a PHP script, the Zend will call the required C code and return a value as though it a function written in PHP was called. It something is not achievable using PHP alone an extension can be written. There PECL and PEAR are two repositories of PHP extensions.

2.4.1 Dynamic Typing

PHP has no static type checking or variable declaration. Variables can change between types during run time. For example a string can be changed into a array:

```
$variable = '123:ABC';
$variable = explode(':', $variable);
```

$variable is assigned a string value, it is then assigned the value returned from explode which will be an array containing two strings. Dynamic typing may not be a desired feature when implementing the CMIS domain model. Code written in PHP can suffer from a lack of clarity because CMIS attempts to assign structure this could be an issue. When releasing the implementation documentation will play an important role as the intended functionality may not be as clear.

Hack: Statically types PHP

Hack is a language created by Facebook that is derived from PHP[4]. Hack provides explicit typing on parameters, class member variables and return values. In comparison PHP only provides basic type hinting for parameters. PHP 5 introduce type hinting allows for parameters to be limited to objects, interfaces, or arrays (introduced in 5.1)[5]. It does not allow for hint-

[4]http://hacklang.org/

[5]http://php.net/manual/en/language.oop5.typehinting.php

ing at the level of scalar types such as int, string, Boolean, With Hacks' Type Annotations and the ability to specify the return value the methods returned would aide in the clarity of the methods within the CMIS domain model.

Hack also allows for more strictly typed parameter, ensures that only integer values are stored as integer properties of a CMIS object. To do this in PHP it would be necessary to add code to check the types at run time.

Hack also introduces generics and more built in structured data types (collections and pairs), these are similar to those found in Java. PHP has arrays, though the array object provided by PHP is not necessarily the traditional concept of an array. PHP accept mixed values and are stored as an ordered map, they can be used as arrays, vectors, hash tables. Keys used for arrays are cast to integers where possible, the string "1" and the float *1.5* are both treated as the integer 1. Non-decimal string values are used as keys as is.

OpenCMIS makes use of Java's generics and collections. Properties which may contain multiple values, these are stored as a collection making use of generics to ensure properties values match the property type.

Hack has developed because Facebook saw a need for these features. Implementing CMIS these features would be desirable, as they would allow for more descriptive code.

Eshkevari et al. [2015] have investigated how some PHP applications use dynamic typing and the changes that would be needed to enforce static typing. They found that less that 1% of assignments used dynamic typing. A large proportions of cases would relatively simple to avoid by renaming local variables. If static typing offers more understandable code, the flexibility dynamic typing offers may not be worth the cost. Perhaps Hack would be a more suitable platform for implementing CMIS. Hack may become more widely used and a Hack implementation of CMIS may become more desired.

2.4.2 PHP interpreters

The Zend engine is not the only interpreter available for PHP, though it is provided by the PHP group and is widely distributed. There are other methods of using PHP code. Facebook are a company that have built technologies using PHP and one of their interests has been optimising performance of PHP. They have released two open source projects perusing this.

Facebook have developed *HipHop for PHP* (HPHPc), which is source code transformer. "HipHop programmatically transforms your PHP source code into highly optimized C++ and then uses g++ to compile it" Zhao [2010]. They state that the use of HPHPc reduced the CPU usage on average by about fifty percent. This project has been succeeded by *The HipHop Virtual Machine* (HHVM) Paroski [2012]. Which was released when it overtook HPHPc in terms of efficiency. Stating the performance of HHVM continuing on an upward trajectory. See Figure 2.3.

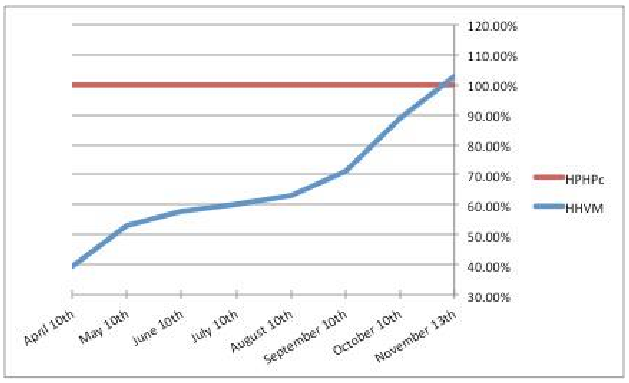

Figure 2.3: Efficiency of HHVM relative to HPHPc as of November 2012.

HHVM could offer an interesting point of comparison when evaluating the performance of the implement of CMIS in PHP. The following is a description of how HHVM works from it homepage.

> Rather than directly interpret or compile PHP code directly to C++, HHVM compiles Hack and PHP into an intermediate bytecode. This bytecode is then translated into x64 machine code dynamically at runtime by a just-in-time (JIT) compiler. This compilation process allows for all sorts of optimizations that cannot be made in a statically compiled binary, thus enabling higher performance of your Hack and PHP programs. [Facebook, 2015]

As of 19th of November 2014 HHVM was added to Archlinux's repositoies[6]. Archlinux is a distribution of Linux which attempts to stay on the "bleeding edge" and will tend to adopt packages earlier than other distributions. At the time of writing HHVM is not available from the main Debian or Ubuntu repositories, but the project does provide a repository themselves. HHVM being added to Arch Linux's repositories is a sign the project is gaining traction.

HHVM is also only available for x64 machines, this may limit the cases in which it can be used. It is also worth noting that Hack is written for use

[6]https://projects.archlinux.org/svntogit/community.git/log/trunk?h=packages/hhvm

in HHVM, it is not usable in all cases where PHP is used.

The performance of the Zend Engine was also improved with the addition of features like Zend Opcache, that have been added since HHVM initial release.

2.4.3 CMIS and PHP

Implementations of a PHP CMIS server are not available, at least not widely. A stub of server is available on github[7]. This has not been updated in the past two years and has the note that it is under “Heavy Development”. This project should results in a usable implementation in PHP.

2.4.4 Front Control Pattern

The front controller pattern is a design pattern that is commonly used in PHP applications. The design pattern is not exclusively used in PHP but is a consideration for this report. Wordpress and Drupal are two example of PHP CM systems that make use of the front controller pattern.

The patterns uses a single point of dealing with requests. This affects the URL structure of the site, for example Wordpress sites use the following structure:

```
example.com/index.php?p=49
```

The query string contains the field emphp with value. The index pages handles most requests, setting this emphp value indicates the value of post request. Wordpress uses different parameters to indicate what is being requested: emphm, request the archive of posts from a month; emphcat, is used to get posts from a category. Wordpress can hide this underlying structure with the use of rewrite rules, with the cooperation of the web

[7]https://github.com/maddingo/php-cmis-server

server URLs can be written so that they are intrepeted by the server as a query string but appear to the user as “nice” descriptive URL. The CMIS implementation is going to be built on the presumption that url structure may be using the front control design pattern.

2.5 Similar Standards

The CMIS standard is not alone in attempting to address the issue of interoperability.

2.5.1 JCR

Content Repository API for Java (JCR) is a specification for providing generalised interface to content repositories [8].

Like CMIS, JCR is a standard that applies a structure to a content repository[9]. Some of the features that it offers are also available from by CMIS, these include: a query language, versioning, access control. JCR differs in that it does not provide access to the repository remotely. It is a set of objects that must be defined within the content management system. This means that code written for one JCR compliant CMS can be used in other JCR compliant systems. In effect JCR dictates how a content management system should be written, whereas CMIS define how the information should be made available.

The project Apache Jackrabbit provides a reference implementation[10]. It is possible to use JCR to make a system CMIS compliment. Part of Appache Chemistry aims to “bridge between the CMIS and JCR stan-

[8]https://wiki.jasig.org/download/attachments/22940141/JCR_iECM_CMIS_JASIG_final.pdf
[9]https://jcp.org/en/jsr/detail?id=283
[10]http://jackrabbit.apache.org/

dards”[11]. Allowing a CMIS client talk to JCR server.

PHPCR

One of the differences between CMIS and JCR is that JCR is language specific. PHP Content Repository[12] (PHPCR) is and adoption of the JCR to PHP. This project is interesting to see a desire for this type of standardisation within the PHP community.

[11]http://chemistry.apache.org/java/developing/repositories/dev-repositories-jcr.html
[12]http://phpcr.github.io/about/

Chapter 3
Design and Implementation

The objective of this dissertation is to minimises the work required to make a CMS written in PHP CMIS compliant. Taking an approach similar to OpenCMIS, hiding the workings of the bindings allows a CMS developer just implement the services as necessary. Early during development to following requirement we decided upon:

Be usable with multiple CM systems

The library should be able to work with range of open source CMSes.

Have similar dependencies to typical CMSs

The library should attempt to not require any extension that are not commonly available in situations where CMS are in use. This is to minimise the effort in using the library.

Though it is possible to extend PHP with extensions written in C, this would result in some level of type checking and benefit from being compiled. Implementing the CMIS data model in a low level language would present more challenges. Requiring the installation of extensions, this would require more control over a web server to install the library. Implementing the library in PHP with the libraries commonly installed would reduce the complexity of the task.

Then with this CMIS implementation applying it to a PHP CMS.

3.1 Current State of the Implementation

The implementation is not a full CMIS implementation. 42 out of the 64 data objects are implemented, some objects like the constream and extension element are not yet done. However of the interfaces for the data objects and definitions have been generated from the Java interfaces, the detail of this follows later in this chapter. All services have a basic implementation that raise a *NotSupportedException*, it is necessary that these are implemented on a per system basis. Providing default implement, that raises an exception, ensures that only the user of the library can implement only their desired functionality.

The data objects implement were chosen so that the getRepositories(), getRepositoryInfo(), getChildren(), and getObject() could be implemented.

The enumerated types mentioned in the standard have been implemented, though there is more to be said about the support for enums in PHP. The exceptions covered by the standard have also been implemented.

The implementation uses a factory design pattern, so allow for user implemented services to be passed to bindings. An additional service is declared to add some that is required by the bindings.

An attempt at the Atom binding was made. 3 of the 18 resources have been implemented. The getRepositories() and getRepositoryInfo() services are accessed through the Atom Service document. getChildren() is exposed through the Folder Children Collection, getObject() is exposed through the Object Entry.

3.2 Implementing CMIS in PHP

The implementation proposed varies from the OpenCMIS implementation in dealing with the service objects. OpenCMIS provides interfaces for each of the services according to their grouping (Repository Services, Navigation Services...) It then introduces a *CmisService* is an interface which extends each of the other interfaces for services. A user of OpenCMIS must then implement one object that provides all the services methods. This CmisService also adds some extra functionality, because the standard acknowledges that the bindings do not necessarily match the services.

For the implementation the decision was made to declare the groupings of services as objects that provide default implementation. It is not possible for an object to extend multiple other objects, the CmisService includes instances of the other services as member variables.

A factory design pattern is used so that the user of the library can implement their own Service with the repository specifics. The user must extend the service objects and the CMIS service factor, so that it sets their service objects as the members of CmisService.

3.2.1 CMIS Domain Model

The decision was made to attempt to match the implementation of the OpenCMIS data model. OpenCMIS is used within the Alfresco ECM, it is functional. Copying the objects declared within OpenCMIS ensures that the necessary functionality is covered. Also there is the benefit that any developers familiar with the OpenCMIS implement will be able to carry over knowledge of the structure of the reference implementation.

In the OpenCMIS packages org.apache.chemistry.opencmis.commons.data and org.apache.chemistry.opencmis.commons.definition there are a total of 96 interfaces declared. They do not contain any implement of methods and have similar structure. Taking advantage of these two facts a ruby script was written to iterate over the files in a given directory and generate valid PHP interfaces from detected java interfaces. Because there is no implementation within interfaces the code is simple enough that it is possible to automate the process of changing the code to be PHP. Because the code followed a structure use of regular expression was an viable option.

The approach for generating the PHP interfaces was as follows:

For each Java file found in a directory or subdirectory:

- Search for interface declaration if found then:
- Open new PHP file in output directory.
- Write PHP open tag and namespace to PHP file.
- Write licence text from from Java file to PHP file.
- Find imported packages in Java file and write list of required packages as comment in PHP file.
- Write Javadoc comment block to PHP file.
- Manipulate interface declaration, removing any use of generics, so that it is valid PHP. Write this to the PHP file.
- Keep record of any extended interfaces, so that it is possible to ensure any dependencies are satisfied when scan is complete.
- For each method found manipulate into valid PHP. Write Javadoc and method declaration to PHP file, also add comment indicating return type.
- End interface with “}” and close the PHP file.

Once all files are scanned check all dependencies (anything that an interface extends) put message to standard i/o indicating unsatisfied dependencies.

3.2.2 PHP Namespaces

Namespaces are a mechanism to encapsulate parts of code written in PHP. It was introduced in PHP 5. This is necessary to use namespaces when implementing a library that is to be used within an application, such as the CM systems the CMIS implementation will be used in. Because the CMIS library is intended for used within a content management system it is important to avoid name collisions. It also allows for the logical grouping of the objects and methods of the CMIS implementation.

It is possible that the CM system that the library is being used in might have objects declared that could conflict with the objects declared in CMIS. The CMIS that the library is being used in could have its own “VersioningService” or “ObjectData” object declared. PHP does not have support for function or method overloading. The objects should all be within a particular namespace to avoid the potential for conflicts. Without the use of namespaces it is necessary to use longer object names and prefix objects with an identifier to avoid name conflicts.

The name given to the implementation for CMIS in PHP is “Elaphus CMIS”, this is in reference to the Cervus Elaphus, or the Red Deer. A specie of deer whose population fell to low levels during the 20th century in Ireland. With protection and management the population has grown, increasing by over 500% between 1978 and 2008. Aiming to draw a parallel to need for standardisation of CM system and the hope that through management interoperability could become less of an issue.

The namespace used in this project was: Elaphus\Cmis. The interfaces defined with the implementation are declared within Elaphus\Cmis\Interfaces This decision was made so that the interfaces would be logically grouped and when an interface is used for type hinting it is clear the type specified is not an instantiable object.

3.2.3 Types

Type Hinting

Elaphus CMIS makes use of the ability to constrain parameters to descendants of interfaces. If a parameter is specified to be an object or interface the method will accept any object that is a descendant of the specified parameter. This ensure that the parameters passed will be an object with the class methods needed for that function.

The object data class uses private variable to store the properties associated with that object. The setter method for the properties array uses type hinting to ensure the parameter passed implement the *getProperties()* method.

It is not possible to ensure total type consistency from type hinting alone. As mentioned in the chapter 2 Hack and Java provide the concept of generics, this can be used to ensure that a hash map only contains objects of a certain type or ancestor. The level type hinting provided by PHP will only allow for parameters to be array (which are more similar to hash maps) and not provide a mechanism to specify what the array can contain. The implementation in Java allows for the all of the properties to be set by passing a Collection containing PropertyData.

```
void addProperties(Collection<PropertyData<?>> properties)
```

While equivillent declareation PHP may look like:

```
function addProperties(array $properties)
```

Looking at this method declaration there is no indication regarding what should pass in the *$properties* array. The return type of functions cannot be indicated in PHP, this can lead to a lack of clarity in the implementation. This means that the documentation for Elaphus CMIS will play a much more important role. The library is intended to allow the user not worry about the data model of CMIS. The lack of clarity discussed here is undesirable, as the objects declared provide less information and could lead to confusion when use Elaphus CMIS.

The type hinting used should not prevent any users from extending objects provided by Elaphus CMIS, if they want to alter it for the specific requirement of their system.

Type Evaluation

Atom print word ‘true’ and ‘false’ for boolean values. Adding a Boolean variable as the content of a DOMElement will result in the value 1 for true and unset if the value is false. There would substantial increase of path complexity as a result of using if/else statements. Increase is path complexity is more of an issue when using interpreted languages like PHP compared to compiled languages. PHP provides var_export function which can be used to get the strings ‘true’ or ‘false’ from a boolean value. However this can only be used to evaluate booleans value, if use an a variable containing a string this would output the string value surrounded by quotation marks. This means that:

```
  $value = "abc";
  var_export($value, true)
```

The call to var_export will return the value “abc” as a string, whereas if $value is a Boolean the string true is outputted, without surrounded quotation marks.

Lack of Native Enum type

A limitations of PHP is the lack of enum type. CMIS defines some enums which are used both as service parameters and as values returned as part of services. For example the *getRepositoryInfo()* return *Enum supportedPermissions*, all bindings will represent this value as the string “basic”, “repository”, or “none”. The Java implementation provides an instantiated objected with an enumerated value, a call to the *toString()* or *name()* will return the name of this enum constant[1]. This is the value used to represent this in the bindings.

There are various solutions for achieving behaviour in PHP. There are several projects on GitHub providing “Java style” enums. For this project an implementation of enums was from a GitHub project that provided the desired functionality and had the highest rating on GitHub. The project was released under the MIT licence.

An alternative approach commonly used for enums in PHP is the use of an abstract class with the enum values declared as constants within the class. This method is mentioned in Hacks documentation, it is a simpler approach. However it will not allow for type hinting as instance of the enum type in method parameters. The abstract class only provides a mechanism to refer to a value defined elsewhere[2].

This is not a major barrier in implementing CMIS in PHP, but it stands as

[1] http://docs.oracle.com/javase/7/docs/api/java/lang/Enum.html
[2] http://docs.hhvm.com/manual/en/hack.enums.php

a reason to question how language independent the standard is. Having parts of the implementation dependant on non built-in functionality could mean that the functionality is not intuitive for developers using the library.

3.2.4 Atom Binding

Elaphus CMIS's implementation of the bindings is not the concern of the average end user. The library is written to hide the implementation of the bindings so that the end user can focus on the implementation of the services. The structure of the Atom binding will likely be reusable for other bindings. Implementations will declare their own end point for bindings. The user will need to handle the repository specific binding selecting. If a request is made to the atom binding the user will simply need to instantiate an *Atom* object and pass a CMIS service to it. As follows:

```
$creator = new MyCmisCmisServiceCreator;
$cmisService = $creator->createCmisService();

if ($_GET['cmis'] == 'atom') {
  $atom = new Elaphus\Cmis\Atom;
  $atom->request($cmisService);
} elseif ($_GET['cmis'] == 'browser') {
  /*  Browerser binding stub */
} else if ($_GET['cmis'] == 'ws') {
  /* WS binding stub  */
} else {
  /*  Display welcome to CMIS page */
  echo 'Elpahus CMIS Wordpress Server';
}
```

The Atom object will handle exception handling and determine the resource to return by using a dispatcher mechanism.

The Atom binding is based upon Atom (RFC 4287) and the Atom Publishing Protocol (RFC 5023). The binding was chosen as it an XML based binding similar to the Web Services binding. Wordpress natively provides an Atom feed of posts this offers a point of comparison.

AtomPub uses a RESTful architecture. This means that it makes use of HTTP actions hyperlinks as an engine for discovery. This means the Elaphus CMIS Atom binding needs to be aware of the capabilities of the repository. The standard states that the representation SHOULD provide links to implemented service that are relevant in a resource representation. CMIS does not include a mechanism for declaring what services are supported, some services may not be supported by repositories. An additional (non-optional) service that states what operations are support could be added to the standard.

The CMIS standard extends the Atom and AtomPub to include some CMIS specific features. Adding XML namespaces for describing the parts of the CMIS objects. These include elements for CMIS object properties.

Dispatcher

A dispatcher mechanism has been implemented that maps the HTTP action (GET, POST, PUT, DELETE) and resource identifier to the associated resource as a PHP object and method. This is done using a multidimensional PHP array, the first index of the array maps to HTTP action, while the second identifies the resource. The returned value is a string that corresponds to the object/method to call. For example the HTTP GET request within no action declared, corresponds to the method for displaying the Service Document.

Generation of XML documents

Aspects of PHP make it well suited for a CMIS implementation. PHP is primarily used a server side scripting language. Many features are built in to aide in this as a result. PHP comes with tools for generating and handling XML documents and dealing with JSON. This will be advantageous when implementing all of the bindings.

When generating the XML documents there were two approaches:

1. PHP is used for “embedded HTML” It would be possible to generate XML by “printing” using echo to output to the response. Care would be needed to ensure a valid XML is created. This would not lead to ideal exception handling.
2. PHP 5 provides the DOMDocument object which can be used to generate and manipulate XML. Using this approach ensure the well formed XML is created. The methods that generate XML document can then return a DOMDocument objects and nothing is printed until all methods that could cause an exception to be raised have been called. This allows for cleaner exception handling as nothing has been printed to the response within the resource methods.

Use of the DOMDocument was chosen.

3.3 Using the PHP implementation in Wordpress

3.3.1 Applying the CMIS data model to Wordpress

Wordpress started as a blogging system, but has evolved to be used as full content management system. The behaviour of aspects can be changed through plugins, these can modify the many aspects of a Wordpress site. The CMIS implementation will be plugin to Wordpress.

All document objects within wordpress are stored as “posts”. The following are the default post types: Post, Page, Attachment, Revision, Navigation Menu. Custom page types can be defined.

To apply the CMIS structure to Wordpress the approach used was to:

1. Look at the Wordpress Dashboard, this is the “admin area” of the site and provides the user interface (UI) for for creating and managing the site.
2. Look at the underlying database structure, this will provide a means to discover the data associated with aspects of the site and what can be made to CMIS properties.

3.3.2 Wordpress Posts from the UI of Wordpress

Looking at the WordPress dashboard. The following options are available to users for creating posts.

Posts These are typically and most commonly used by blogs, these are stored as under the “post” post-type. They are displayed in reverse sequential order, providing a timeline of posts with the most recent posts at the top. Posts can be categorised, possibly to multiple categories, this would work in conjunction with CMIS multi-filling capability. If a post is not assigned a category it will placed placed into the “Uncategorized” category. This is a default category provided by Wordpress, however it can be renamed. A post must have at least one category.

Pages These are typically used to represent “less time-dependent” information of a website. Like posts these are HTML data. However they are not filed under categories but instead use a weaker form of hierarchy. They are organised in a hierarchical structure using a parent attribute. This is a relationship from a post to another post of the page type, this is not required.

Media Which includes Images, videos as well as pdf or other files uploaded to the server. These are stored under the “attachment” post-type.

Additional default post may be less apparent to users of Wordpress, both to the site users and owners.

Navigation Menu Allow for custom navigation menus into a built into a theme. These are made using the Wordpress Dashboard are more theme specific.

Revisions These include: auto-saved copies of pages/posts and saved drafts of posts/pages. They create a history of published update to a post. As changes are made the original wp_post entry is updated, an a record of the state is created. The id of the original post will not change, this fit that the post id can be used to generate the CMIS concept of an immutable object id.

3.3.3 Wordpress Posts Database structure

There are two tables containing information about posts. When using Wordpress's default table prefix (wp_) these are called: wp_post table and wp_postmeta.

wp_post is a table with 23 attributes. The primary is an *AUTO_INCREMENT* integer value. The table contain values used as foreign keys, foreign key constraints are not enforced by the dbms instead are managed at the application level. Many attributes may be left null depending on post -type. There are attributes that are always set such as: post_author, which is a foreign key to the ID of a wp_users table tuple; a GUID, which is contains the URL for that item; post_type. Some attributes are only set for particular post types: an attachment will not have any post_content set, while an post or page will not have a post_mime_type set.

Table 3.1 shows the mapping of CMIS properties to values from Wordpress table entries. When a values are tables are indicated by <*table name*:*attribute name*>. When a table other than wp_posts is used a foreign key from the wp_posts entry is used to find the corresponding value from the other table.

Property	Description	WP_value
cmis:name	Name of Object	<wp_posts:post_name>
cmis:description	Description of the object	
cmis:objectId	Id of the object	post:<wp_posts:ID>
cmis:baseTypeId	Id of the base object-type for the object	cmis:document
cmis:objectTypeId	Id of the object's type	cmis:document
cmis:secondaryObjectTypeIds	Ids of the object's secondary types	
cmis:createdBy	User who created the object	<wp_users:display_name>
cmis:creationDate	DateTime when the object was created	<wp_posts:post_date>
cmis:lastModifiedBy	User who last modified the object	
cmis:lastModificationDate	DateTime when the object was last modified	<wp_posts:post_modified>
cmis:changeToken	Opaque token used for optimistic locking and concurrency checking	
cmis:isImmutable	Defines if the object can be modified.	
cmis:isLatestVersion		Requires Calculation
cmis:isMajorVersion		true
cmis:isLatestMajorVersion		Requires Calculation
cmis:isPrivateWorkingCopy		Requires Calculation
cmis:versionLabel		Requires Calculation
cmis:versionSeriesId		
cmis:isVersionSeriesCheckedOut		
cmis:versionSeriesCheckedOutBy		
cmis:versionSeriesCheckedOutId		
cmis:checkinComment		
cmis:contentStreamLength	Length of the content stream (in bytes).	
cmis:contentStreamMimeType	MIME type of the content stream	<wp_posts:post_type>
cmis:contentStreamFileName	File name of the content stream	<wp_post:guid>
cmis:contentStreamId	Id of the content stream.	

Table 3.1: Mapping of CMIS Document Properties to Wordpress Posts

Some properties may be unset, for example properties relating to content streams may left empty when a document does not have an associated content stream. Similarly objects do require values in the secondaryObjectTypeIds property.

Properties related to versioning have the following note in the standard:

> The repository MUST return this property with a non-empty value if the property filter does not exclude it. Version property values are repository-specific when a document is defined as non-versionable. [TC, 2013, p. 39]:

To determine if a Wordpress post is the latest version a it will be necessary to determine if any revision posts with the parent_page value set to the post id are drafts or are published. To determine if a revision is a draft or has been made the post_status tuple will need to be checked. A similar method is necessary to determine the lastModifiedBy, using the aurother of the "published" revision.

Wordpress has basic versioning built in, it does not include the concepts such as a private working copy or checking in/out documents.

wp_postmeta contain additional information about posts, information is stored as key value pairs. This is where some meta data is stored about media for example. The key "_wp_attachment_metadata" is used and information about alternate image sizes and camera metadata is stored as a string. No file size is stored however, determining the content stream length will need to be by querying the file from the filesystem.

There is no property for the HTML markup of page, an additional HTML type property was included in documents. This was called "wp:post" and contains the markup as it is stored in the database, a fragment of HTML markup. The standard states that value of a HTML property may contain fragments of HTML and do not need to guaranteed validity. This property key was chosen for the implementation of the plugin built on Elaphus CMIS, if other plugins are to implement CMIS this could be point at which implementations could vary. This could affect interoperability between CMIS implementations.

3.3.4 Wordpress Category Database structure

Another wordpress object that needs to be mapped to the a CMIS object type is categories. Categories function like folders for posts of the post type. They have a hierarchical structure, categories may have a single category as a parent.

Table 3.2 shows a suggestion for the mapping of CMIS folder properties values from Wordpress value. While the mapping a post to a document presents few challenges, Wordpress categories are not as equivalent to CMIS folders.

Wordpress stores information about categories across 3 tables:

wp_terms Contains the definition of Wordpress categories, the id, name and slug (name for use in URLs).

wp_term_taxonomy Provides more information about entries in the wp_terms table, a description, their parent (foreign key relationship) and number of posts.

wp_term_relationships Contains the post ids (*object_id*) and term ids (*term_taxonom_id*), this is used to place posts into categories.

No information about the creator of categories otr changes in their history is stored in the database. This means there is no source of information for this properties. It may be possible the for CMIS wordpress function to record changes to categories itself. However as CMIS is intent to be an abstraction layer this seems to go against the intention of the standard. The standard is not written to change the structure used to store information in a CM system. It does not seem desirable to have the CMIS implementation, or a plugin providing CMIS to affect the database of the system.

Property	**Description**	**WP_value**
cmis:name	Name of Object	<wp_terms:name>
cmis:description	Description of the object	<wp_term_taxonomy:description>
cmis:objectId	Id of the object	cat:<wp_terms:term_id>
cmis:baseTypeId	Id of the base object-type for the object	cmis:folder
cmis:objectTypeId	Id of the object's type	cmis:folder
cmis:secondaryObjectTypeIds	Ids of the object's secondary types	
cmis:createdBy	User who created the object	
cmis:creationDate	DateTime when the object was created	
cmis:lastModifiedBy	User who last modified the object	
cmis:lastModificationDate	DateTime when the object was last modified	
cmis:changeToken	Opaque token used for optimistic locking and concurrency checking	
cmis:parentId	Id of the parent folder of the folder	folder:root folder:posts cat:<wp_terms:term_id >
cmis:path	The fully qualified path to this folder	Requires Calculation
cmis:allowedChildObjectTypeIds	Id's of the set of object-types that can be created, moved or filed into this folder.	cmis:document...

Table 3.2: Mapping of CMIS Folder Properties to Wordpress Categories

3.3.5 Site structure

To generate id for objects the id use by Wordpress are useful. There are two tables that store information about objects. To differentiate the ids a prefix was used, for entries from the wp_post table ids took the form "post:*id*". For categories the prefix used was "category:*id*". When implementing the services the repository specific implementation is written to look at value before the colon and make the relevant Wordpress functions based on this part of the object id.

Wordpress does not structure its sites in a strict hierarchy as CMIS does. It is necessary to create some CMIS specific folders under which the items within a Wordpress site can be presented. This is another point at which implementation of CMIS for Wordpress could potentially diverge.

For this implementation a root folder was created with the id “folder:root”. This had 3 children folders:

folder:posts This is the access point for post type posts. All children this folder are categories, only categories that do not have a parent relationship set in the wp_term_taxonomy table are listed under folder:posts. categories will have children that are either posts or other categories. When generating the list of children is necessary to iterate over all categories, this is a limitation of Wordpress as parent relationship is only mentioned in child categories.

folder:pages provides a list of all pages, as they do not have a hierarchy in the same folder manner this only access point for pages.

folder:media access for media, this operates similarly to folder:pages.

Chapter 4
Evaluation

To evaluate the performance of the CMIS server the following metrics will be used: the CPU load on the server and the memory usage of the server.

These will be measured under different workloads. A simple client will be written that requests random documents from a data set. The client will attempt to request pages at varying page requests per second. The client should not do any work with the results of requests and try to closely match the desired number of page requests per second.

It is not the objective to evaluate the performance of the web server or of the interpreter used, however benchmarking the code base running with different interpreters is an objective. There are layers of caching that cause skewed data. It is possible that either the application server could return a cached version of a page, or that the interpreter could avoid having to regenerate the document. To avoid the possibility of caching a large data set should be used. MySQL will also cache the results of SELECT statements. If the statements are the same it will not need to run the query again.

4.1 Generation of a the dataset

It is necessary to populate the Wordpress repository with large enough dataset so that the server will need to generate a different for requests and cannot cache the generated document. To achieve this 12,000 Wordpress posts were generated. A short php script was used to generate post using content from lorem ipsum generator. The content of the posts had a fixed number of paragraphs of similar length, the content was randomly generated and varied from post to post. The title of the posts varied by having a number appended to the word “Ipsum”. The posts were generated so that they would be in order, this meant that the post IDs were sequential and the client could easily pick a post by generating a random number within the range of post IDs.

4.2 Details of the machines used for testing

When evaluating the CPU usage of the some of the features CPU will need to be turned off. In particular CPU frequency scaling, Intel CPU can come with “Intel Speedstep”. It is important that this feature is turned off so that the base workload of the server can be measured without aiming for a moving target. When the processor is under a lower workload it is switched to a slower clock frequency, this allows for power conservations however this would cause unreliable results.

For the benchmarks performed it was necessary to use a low performance machine (when compared to a scaled server machine) as the server; this will be referred to as device 1. This device had an Intel Atom Processor N2600 and 2GB of ram and no swap space (room allocation on the hard disk for pages). The typical use case for a device with this processor is not as a server, it was chosen because it was possible to disable Intel SpeedStep in the BIOS of this machine. When using a more powerful machine a greater rate of page requests per second would be expected. The clock rate of this device was limited to 600 MHz.

Another device was also used. This machine had a Intel Core i3-4000M CPU with 8GB of ram; this will be referred to as device 2. The clock rate of this machine ranges from 800 Mhz to 2.4 GHz, however due to limitations of the BIOS it is not possible to turn off Intel SpeedStep. If this machine was as the server, requests would be processed more quickly.

Both machines used an x86_64 install of Arch Linux as the operating system. The web server used was lighttpd version 1.4.35-1. The version PHP used was 5.6.8-2. The version HHVM used was 3.7.0-1. MariaDB was used to provide mysql server, the version used was 10.0.17-2. Wordpress version 4.2.2 was used. The version of the Linux kernel used was 4.0.1-1.

4.3 Means of measuring server performance

It is necessary to measure the CPU and memory usage on the server. Requests made to the server are handled by multiple processes: the web server, PHP CGI processes, MySQL, HHVM. A means of measuring all of these process is it to look at the total CPU and memory usage of the system. This means that it is necessary to limit the other processes running on the server, as these would be a source of unwanted activity on the machine.

This data should be obtained during sessions of requests. A bash script was used to obtain these values, the script would take a reading and store that as a text file and sleep for a period of time.

4.3.1 CPU usage

Details about CPU usage was obtained from “/proc/stat”[1]. This contains information about the time the CPU has spent on different tasks since boot. The time spent on user and system process is measures during a session of requests. Since the processor time include information from boot the initial value obtained should be subtracted from all later readings.

Polling and storing of the values for CPU usage will take up processor time, a baseline of the script running without the server handling any requests will be needed.

4.3.2 Memory usage

The memory usage was measured using the “free” command[2] Since no swap was allocated on the device used as a server only the physical memory was checked.

4.4 Design of a simple client for benchmarking

It was necessary to design a client that would do minimum work and attempt to closely match a desired request rate. A client was written in Ruby with three different methods for getting a set request rate.

1. A client built under the assumption that the time per request would be negligible. This client would make a set number of requests with equal time between requests. This avoids the overhead of checking the time taken for each request.

[1] http://www.linuxhowtos.org/system/procstat.htm
[2] http://linux.die.net/man/1/free

2. A client that makes a fixed number of request but calculates the time taken for a request so that the frequency of requests can be varied. This method should better match the request frequency, if the time taken for a request is not negligible.
3. A client that runs for a set time and varies request frequency to match the desired frequency. This has the additional overhead of checking total run time of session of request.

To determine which strategy would be the most effective, each client was run 100 times attempting to make 100 requests per second over a 30 second period, with a 10 second break between each session of requests. The time and number of requests made per session was measured. It is important to consider whether the request rate matches the desired request rate. As well as running for the required time, this is so that the data measure correlates to workload. This is so that the load on the server can accurately be attributed to the clients requests.

The request were made for random CMIS document entries from a server running HHVM. Both client and server were running on the same machine, the values would not accurately measure the server's performance. Device 2 was used for this test. Non-essential background applications were not stopped, as server performance was not measured. Requests were made to a local machine minimising the time taken to look up the host of the server, the requests were made to ip address "127.0.0.1" They are only suitable for evaluating the approach to take for the client. It is possible to make some observations about the performance of HHVM running for an extend time. HHVM has been stated to have a "warm up period", this is not evident from this test which ran for close to 4 hours with approximately 900,000 requests made to the server. Request time appears to be reasonably consistent, HHVM caches based on hostname of the request. It is possible to speculate that the performance of HHVM will not vary hugely during a test session with a single client.

Table 4.1 shows the results of these series of test. These results show that the client that makes a fixed number of requests with equal time between requests, runs for longer than desired and does not achieve a value near the desired request rate. This client performed the worst and was not used.

	Average Runtime of Session (s)	**Average Request Rate (s)**
Fixed request count Fixed pause	51.37102684	58.41545984
Fixed request count Calculated pause	30.97463337	96.85866851
Varied request count Calculated pause	30.00515986	96.80335313

Table 4.1: Duration and request rate over 100 client sessions

Figure 4.1 shows the results from each of the individual session durations. When doing minimal work to correct for session duration, the clients would run for at least 20 extra seconds. As there is no apparent speed up in response of the server, some of the longest sessions times were towards the end of the test. It can be assumed for the code size used in the test, that HHVM is able “warm up” quickly enough that the effect on results is negligible. The variance of the session time, when there is no correction for pause between requests, is 0.78. This 0.78 variance is over 3000 requests be session, this variance seems small when considering the total number of requests. Multiple factors can attribute to this beyond request

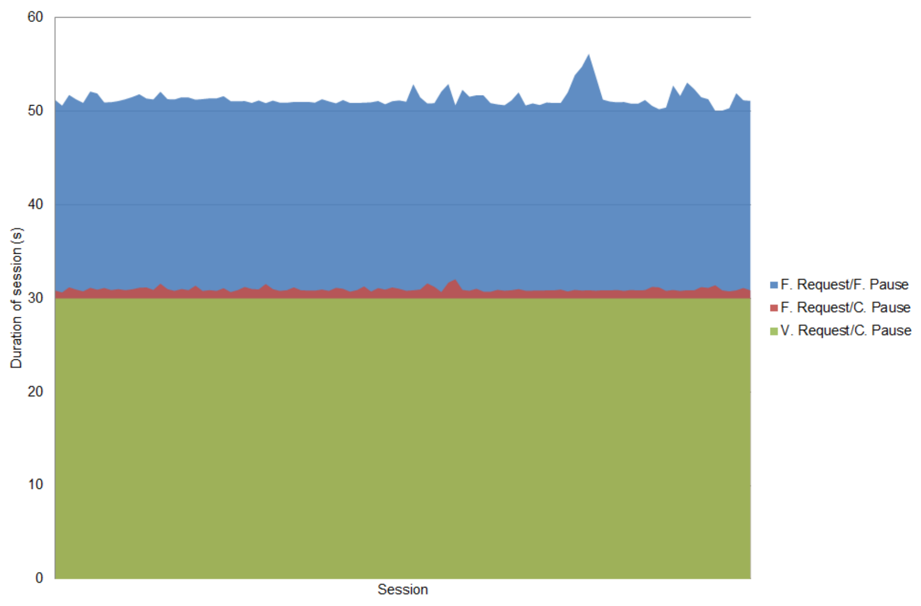

Figure 4.1: Duration of client request sessions

time, noise caused by background process and possible imprecision in Ruby's sleep duration.

A modified version of the client was also made to determine the maximum request rate the server can manage. To achieve this the pause between requests was removed.

All of these clients were designed to make non-concurrent requests. The client would be written to use a thread pooling approach and achieve a page request rate closer to the desired rate. The client would then make concurrent requests, however for this tests the decision was made to only evaluate the server under a non-concurrent load.

4.5 Evaluation of the server running at maximum request rate

The performance of the server was measured while generating the CMIS representation of random wordpress post compared to the web page generated when viewing that page online. The client used attempted to make requests at the highest request rate possible.

The theme used will affect the the amount of work the server must do to generate a post. The web page displayed of a post is theme specific, a theme may include a list of recent posts or links to categories. This information is not required or included in a CMIS document. The theme used for this test was “Twenty Fifteen” the default theme selected in current installations of Wordpress.

The test was run using Zend Engine with Zend OpCache and HHVM using device 1 as the server.

		Time (s)	**Request**	**Req per Sec (s^{-1})**
CMIS	HHVM	600.0338292	4853	8.087877322
	Zend OPcache	600.064528	5330	8.882378063
WP	HHVM	600.3758767	1459	2.430144276
	Zend OPcache	600.0431712	535	0.8916025141

Table 4.2: Achieved max request rate

The server achieved a much lower response rate when generating a the Wordpress page when compared to a CMIS version of a page. This can be expected as the rendered page shows more information from the database and relies on more calls to Wordpress functions. The generating of the CMIS representation of a post is reliant on work in manipulating the XML representation than of the initialisation of objects. As the tests were done using OPcache and HHVM which implements it own caching these repeated methods calls are likely cached. The database requests were more to be the bottleneck as, database queries are a time consuming endeavor.

During these test server resource were monitored. The values for processor time spent during the test are shown in figure 4.2 and figure 4.3. These graphs show a baseline of CPU usage of the server running with no requests being made to it. This will consist of the cost of polling CPU and memory usage as well as any background process that were left running. It is not possible to run this test in complete isolation because the test require an operating system and networking process.

The slope of the graph indicate rate of processor time spent doing work. When the processor is more active, under higher load, less time will be spent idle and the graph will be less steep.

Looking at results from figure 4.2, it shows that less process time is spent when computing a CMIS document object when compared to Wordpress post. The slope of the CMIS representation becomes less steep after a period of time. Before this point there is less difference in the slope of the CMIS and Wordpress representations. Possible reason for this change will be discussed later in this section.

The same test was run with using HHVM. Figure 4.3 shows that more process time was spent when generating the CMIS objects initially. How ever there is a similar change in rate of process use when generating CMIS objects.

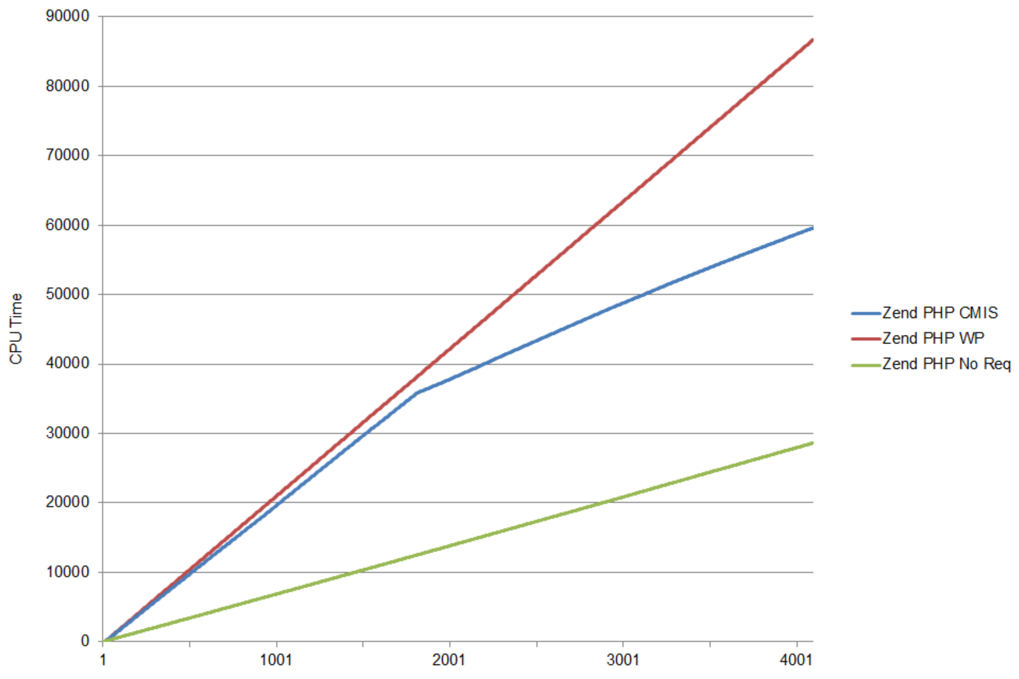

Figure 4.2: CPU time spent during Requests (Zend Engine w/ OPCache)

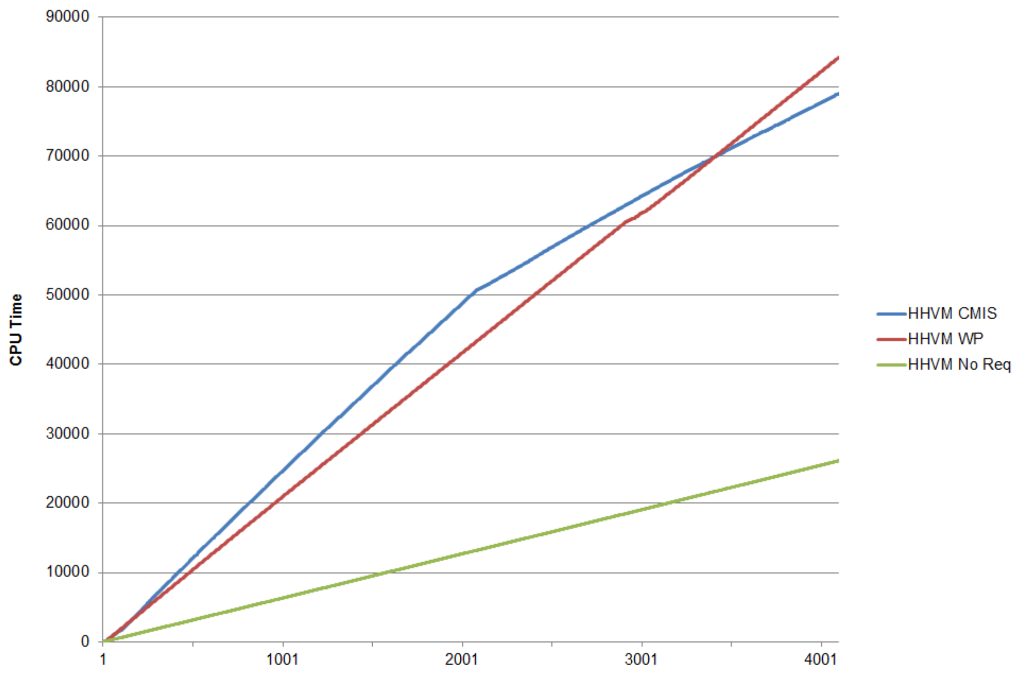

Figure 4.3: CPU time spent during Requests (HHVM)

Speculating on the reason for this change in performance there two reasons:

1. Generation of CMIS objects is less dependent on function/method calls to PHP script. Less time is needed to cache the OP code needed to run the script. If the caching mechanism works similarly in HHVM and Zend OPCache the similarity in results would make sense.
2. A constant between the test is that application server and SQL server are the same. It is possible that due to the smaller response size that caching at one or both of these services causes a gain in performance. Less varied queries are run on the SQL server when a CMIS document is generated. No information about categories or navigation menus is retrieved for example. As the CMIS issues similar request MariaDB could begin operating better after a “warm up” time. Perhaps the application server, lighttpd, begins issues cache hits when asked for the smaller CMIS response.

The memory usage during the test are shown in figure 4.4 and figure 4.5. Again a baseline of the system when no request are being made is shown in green. These values are taken with the necessary daemons running. The baseline memory usage when HHVM is running is higher than when it not. This is expected as HHVM runs in addition to the lighttpd and MariaDB.

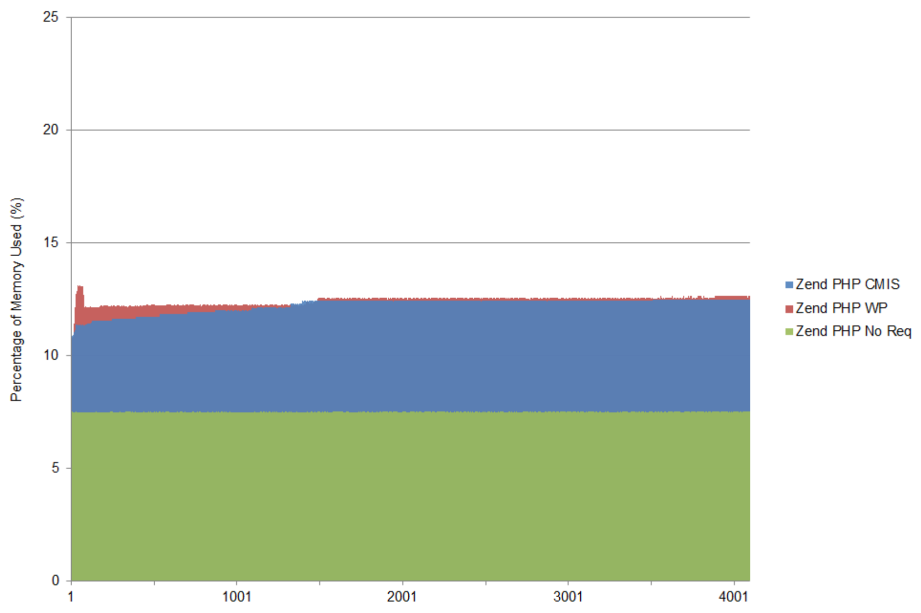

Figure 4.4: Memory usage during Requests (Zend Engine w/ OPCache)

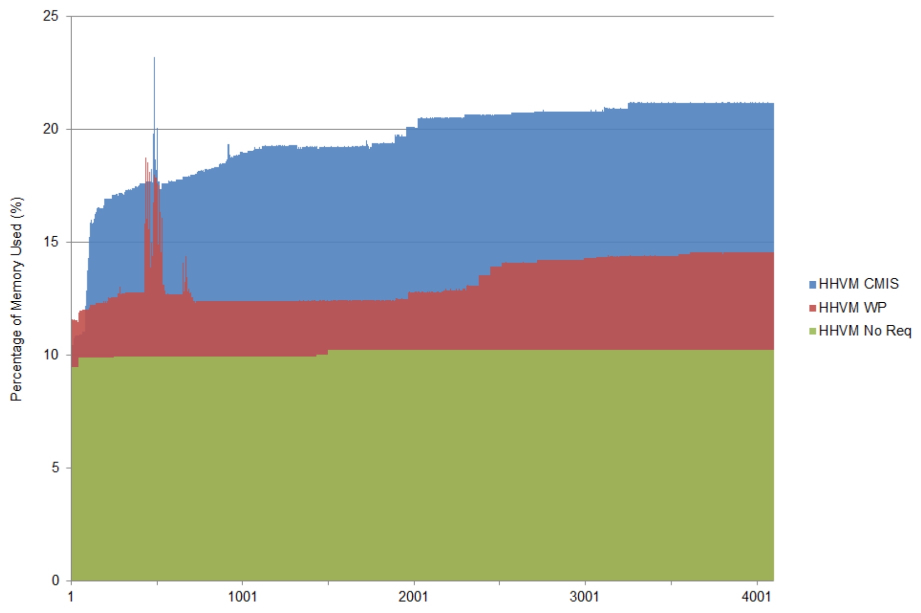

Figure 4.5: Memory usage during Requests (Zend Engine w/ OPCache)

Memory usage show little difference when generating CMIS documents and Wordpress posts in Zend Engine, see figure 4.4. Memory usage when using HHVM shows more a of a difference. When generating CMIS documents HHVM uses more memory, approximately 5% of available RAM. While Zend Engines memory usage remains quite consistent, HHVM increases over time. There could be an issues with HHVMs garbage collection. The generation of CMIS document will result in the instantiation of more objects, this could be the cause of the increased memory consumption.

Chapter 5
Future Work

The implementation of Elaphus CMIS is not yet complete, work should be done to get the implementation to a point at which I could be released as an open source project. When the completion of both: the CMIS data model and Atom binding would be desirable time for release. Documentation will need to be written before release. At that point work would still need to be done to add the Web Services and Browser bindings.

There are potentially interesting area that CMIS could be applied to. Use of CMIS as means to structure data for the semantic web could be an interesting area of research.

Potentially if the CMIS standard was updated to include predefined subtypes for commonly used object types CMIS could function more as an "out of the box" solution. The standard could be extended to include a lookup service to allow for sharing of objects-type definitions.

Chapter 6
Conclusion

A part of the CMIS standard was implemented in PHP. No major barriers were encountered during this implementation. This supports the standards claim of language independence.

Aspects of the standard may have been written with Java in mind. Java's generics and available data structures can aid in the implementation of the CMIS standard. Some limitations of PHP result in Elaphus CMIS possibly being less intuitive than desirable. This will only be determinable once a more complete version of Elaphus CMIS has been released and used by other developers.

Some of the limitations that cause this are: the lack a function/method return type, lack of native enums, weak type hinting. The need for these features has been observed before, Facebook have developed their language Hack that provides these features.

Using the Elaphus CMIS in Wordpress provided largely positive results. The benchmarks performed showed that the generation of Atom CMIS documents is at least comparable to the generation of Wordpress web pages. In some cases the CMIS documents used less CPU time to generate.

The achieved request rates were better in the case of CMIS documents, meaning that as well as causing less of a load the CPU they also could be generated faster.

CMIS may not be an “out of the box” solution to provide interoperability between CM systems. There is room for variations within implementations of CMIS. This is especially evident for CM systems that do not provide a strict hierarchical representation of the data within a repository.

When implementing CMIS in Wordpress there are different approaches available to applied CMIS data model to Wordpress. Interoperability will likely be an issue if different implementations approach this differently. However different implementation of CMIS should will only be able to represent data according to the data model. This means that differences should more easily accounted for than if the two systems are completely different.

Bibliography

M. Achour, F. Betz, A. Dovgal, and N. Lops. Php manual, 2015. http://php.net/manual/en/ Accessed: 2015-05-02.

L. Eshkevari, F. Dos Santos, J. Cordy, and G. Antoniol. Are php applications ready for hack? In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*, pages 63–72, March 2015. doi: 10.1109/SANER.2015.7081816.

Facebook. The jit compiler, 2015. http://hhvm.com/ Accessed: 2015-05-02.

S. Golemon. Extension writing part i: Introduction to php and zend, 2005. http://devzone.zend.com/303/extension-writing-part-i-introduction-to-php-and-zend/ Accessed: 2015-05-17.

U. Kampffmeyer. *Enterprise content management*. PROJECT CONSULT GmbH, 2006.

F. Müller, J. Brown, and J. Potts. *CMIS and Apache Chemistry in Action*. Manning Publications Company, 2013. ISBN 9781617291159.

A. OhAirt. An open localisation interface to cms using oasis content management interoperability services. Master's thesis, School of Computer Science and Statistics at Trinity College Dublin, 2013.

D. Paroski. Speeding up php-based development with hhvm, November 2012. https://www.facebook.com/notes/facebook-engineering/speeding-up-php-based-development-with-hiphop-vm/10151170460698920 Accessed: 2015-05-02.

O. C. M. I. S. C. TC. Content management interoperability services (cmis) version 1.1. Technical report, Organization for the Advancement of Structured Information Standards (OASIS), 2013.

H. Zhao. Hiphop for php: Move fast, February 2010. https://www.facebook.com/notes/facebook-engineering/hiphop-for-php-move-fast/280583813919 Accessed: 2015-05-02.