Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master Thesis

Design and implementation of a Domain Specific Language for defining ECM workloads in elastic cloud environments using TOSCA

Sergey Kukhtichev

Course of Study:	Infotech
Examiner:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Supervisor:	Dipl.-Inf. Tim Waizenegger, Dipl.-Phys. Cataldo Mega
Commenced:	2014-11-01
Completed:	2015-05-03
CR-Classification:	I.7.2

Abstract

Each year the volume of the content produced by enterprises increases by 35%-50%. Most of this information is stored by companies as unstructured data. Organizations implement Enterprise Content Management (ECM) to structure content and to mitigate legal risks. ECM includes strategies and tools for increasing the effectiveness of content management. Using the right ECM components is one of the factors for successful implementation of ECM.
There is a gap between business customers who implement ECM strategies and ECM architects and cloud providers, who create and deploy ECM solutions. On the one side, there are no generally accepted terms, which precisely describe the ECM domain. Different developers of the ECM systems could use the same name for ECM components, which implement different functionality, or they could use different names for ECM components with similar functionality. On the other side, each customer has unique workloads for Enterprise Content Management. ECM architects have to define these workloads and deploy an ECM solution that will fulfill the customer's requirements.
The goal of the Thesis is to develop a Domain Specific Language (DSL) for ECM domain, which will be understandable by business customers, ECM architects and cloud providers. The language should define terms and workloads that are related to the ECM domain. The business customers should be able to create a description of the requirements to ECM solution using a language that they understand. The ECM architect should be able to define the ECM related terms and associated workloads from the customer's description, design a topology, and publish it in a specialized catalog. The cloud provider should be able to map the high level topology to the exact infrastructure.

Contents

List of Figures

Acronyms

AIIM Association for Information and Image Management

API Application Programming Interface

BPEL Business Process Execution Language

BPMN Business Process Model and Notation

CMIS Content Management Interoperability Services

CSAR Cloud Service Archive

CRUD Create Retrieve Update Delete

CRUDS Create Retrieve Update Delete Search

DAML DARPA Agent Markup Language

DMOZ Directory Mozilla

DoD US Department of Defense

DS Directory Services

DSL Domain Specific Language

ECM Enterprise Content Management

ECMS Enterprise Content Management System

eDA electronic Discovery Agent

EDI Electronic Data Interchange

eDM electronic Discovery Manager

FEF Functional Enterprise Content Management Framework

GPL General Purpose Language

IaaS Infrastructure as a Service

IDG International Data Group

ILG Information Lifecycle Governance

IMAP Internet Message Access Protocol

IT Information Technology

KPI Key Performance Indicator

LDAP Lightweight Directory Access Protocol

NAS Network-attached storage

NIST National Institute of Standards and Technology

OS Operating System

PaaS Platform as a Service

SaaS Software as a Service

SAN Storage Area Network

SFTP Secure File Transfer Protocol

SIFS Short Interframe Space

SLA Service Level Agreement

SMTP Simple Mail Transfer Protocol

SSL Secure Sockets Layer

SSO Single sign-on

TOSCA Topology and Orchestration Specification for Cloud Applications

WS Web Services

XML Extensible Markup Language

1 Introduction

Each year the volume of the content produced by enterprises increases by 35%-50%. Most of this information is stored by companies as an unstructured data [BBFRS12]. To structure the content and to mitigate legal risks companies use ECM [LBMCW13]. Content management systems manage the lifecycle of content from creation to disposal. Association for Information and Image Management (AIIM) gives the following definition of ECM: "Enterprise Content Management (ECM) is the strategies, methods and tools used to capture, manage, store, preserve, and deliver content and documents related to organizational processes" [AII14]. Since 2004 Gartner analyses the ECM market and publishes "Magic Quadrant for Enterprise Content Management", where it defines the changes on the ECM market and describes the strengths and weaknesses of ECMS developers. Gartner divides the term ECM in two frameworks: a strategic framework and a technical architecture. The strategic framework allows companies to increase the productivity, take control of their content, enable content-centric processes. The technical architecture, consists of the platform - set of services that provide core ECM functionality, and additional modules that extend platform's functionality. The platform includes the following components: document management, web content management, records management, image-processing applications, social content, content workflow, and extended components [GSC+14].

Examination of the solutions of ECM vendors shows that most of the leaders from Gartner's quadrant provide cloud solutions for their ECM systems. According to the annual International Data Group (IDG) Enterprise Cloud Computer Study that was made across more than 1600 IT and Security decision-makers, investments to the cloud have increased by 19% since 2012 [idg14]. Nowadays more and more companies move their IT infrastructure from on-premise solutions, when the companies maintain their own servers and buy software, to cloud solutions, when the companies rent infrastructure or/and software. By using the clouds the companies could focus on their enterprises and do not need to think about maintenance of servers or about a disaster recovery. Moving ECM solutions to the cloud make new demand to the cloud IT infrastructure. For example, location and disaster recovery policies should be applied. According to the law, some type of data (e.g. personal data) should be stored on servers inside the countries, and that data should be stored in a such way, that it could be restored in a case of the disaster. Enterprises also make demands to the cloud providers. They define additional Key Performance Indicator (KPI)s, like maximum response time or maximum recovery time, that should be satisfied by the cloud provider.

The report is structured as follows: chapter 1 "Introduction" gives the overview of ECM, states the problems that occur between business consumers, ECM and cloud providers and describes technologies, that were used for DSL design. Chapter 2 "Related work" gives the overview of work that was made in the area of the ECM domain analysis. Chapter 3 "DSL development methodology" provides commonality and variability analysis of the ECM domain

and describes an ontology method for the domain analysis, and methodology for implementation of DSL using Topology and Orchestration Specification for Cloud Applications (TOSCA). The results of the DSL development are described in chapter 4 "DSL design results". They describe ECM ontology, concepts and implementation of ECM DSL. The evaluation of DSL was done by creating a ECM solution topology, providing this topology and a questionnaire to the IBM developers and receiving feedback from them. An appendix A contains detailed ECM ontology.

1.1 Problem description

There is a gap between business customers who implement ECM strategies and developers and providers of ECM solutions. On the one side, there are no generally accepted terms, which precisely describe the ECM domain. Different developers of ECM systems could use the same name for ECM components, which implement different functionality, or they could use different names for ECM components with similar functionality. Developers could also introduce the new terms to differentiate their ECM solutions from the competitors. As a result, the customers of the ECM systems couldn't clearly understand the functions provided by the developers of ECM solutions. It is also hard to identify the potential of the new software since the functionality do not necessarily coincide with the names used by ECM developers [Bö14].
On the other side, each customer has unique workloads for ECM. These workloads depend on the different factors [Jen04]. For example, a records management is an important part of ECM strategy. Implementation of this strategy depends on the geographical location of the customer's business. The requirements to ECM solutions for European and US companies will be different and, for example, capabilities developed according European standard will not be accepted by US customers, who implement US Department of Defense (DoD)5015.2 standard.
Moving ECM solutions into the clouds creates new demands for that systems: different types of policies, interoperability and portability of cloud services.
The goal of the thesis is to define the terms of general ECM components, their functional and non-functional requirements, and to design a declarative language for ECM domain. The language should be understandable by the business customers, who want to use ECMS to increase the productivity and mitigate the legal risks, by the ECM architects who design topologies of ECM solutions and by the cloud providers, who deploy ECM solutions. ECM service architects should have the possibility to design ECM topologies and to define an orchestration between ECM components. The business customer should be able to define functional and non-functional requirements and policies of ECMS. Business users should also understand the operations, which each component supports. Cloud providers should have the possibility to map the service topology to the their cloud infrastructure.
The possible use case of ECM language can be defined as follows:

1. The business customer creates the description of requirements to ECMS and sends them to the ECM architect.

2. The ECM architect defines ECM related terms from the customer's description, designs topology, and publishes it in specialized catalog.
3. The business customer fills the properties of the template and initiates the deployment process.
4. The cloud provider defines the mapping of a high level topology to the exact infrastructure. It could be done by developing a special adapter that will transform a high level framework to a low level architecture.

Mapping the requirements of the problem domain to the ECMS architecture is out of the scope of the thesis, and could be implemented by ECM providers as additional adapters that will convert the user's specification into the final ECM solution.

1.2 Fundamentals

This section includes the definition and description of the technologies that are used in the master thesis.

1.2.1 Cloud computing

The National Institute of Standards and Technology (NIST) of the U.S. Department of Commerce defines Cloud computing as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." NIST defines three service models of cloud computing [MG11]:

- Software as a Service (Software as a Service (SaaS)): the consumer uses the provider's application deployed on cloud infrastructure;
- Platform as a Service (Platform as a Service (PaaS)): the provider gives the ability to deploy consumer's applications on its cloud infrastructure;
- Infrastructure as a Service (Infrastructure as a Service (IaaS)): when the consumer has the ability to deploy arbitrary applications (including operating systems) on provider's core resources (networks, storage, CPUs, and etc.).

NIST also defines four deployment models of cloud computing [MG11]:

- Private cloud: the cloud infrastructure is used by one organization;
- Community cloud: the cloud infrastructure is used by several organizations that share the same concerns (mission, policy, security requirements, and etc.);
- Public cloud: the cloud infrastructure is used by the general public;
- Hybrid cloud: composition of Private, Community and Public clouds;

1.2.2 Domain specific languages

DSL is a language that is dedicated to a specific problem area [Gho11]. It consists of semantics and syntax. Semantics define a language vocabulary that includes terms and their meanings. Syntax consists of rules that describe how language terms are related to each other. DSL provides syntax and semantics at the same level of abstraction as a problem domain. DSL users focus only on the problem domain model, and should not think, how this model will be implemented. Typical DSL consists of the problem domain, which is related to the analyzed business, and the solution domain, which refers to the techniques and tool that form the solution for the problem domain model. Implementation of DSL maps problem domain artifacts to solution domain artifacts. For the better understanding between domain experts and solution developers, DSLs share the common vocabulary, which provides the following benefits [Gho11]:

- Vocabulary is used as a glue between modelers and domain experts. Usage of the same terms helps the modeler to speak in the same language with the domain experts.
- Common terminology during tests. Modelers, Testers and Domain Specialists shares the same vocabulary during establishment and execution of test cases.
- Common vocabulary during development allows developers to use the same terms as the domain experts.

DSLs can be divided in two categories: external and internal languages. The internal DSL is developed based on an existed General Purpose Language (GPL). The external language is developed from scratch.
DSLs can be also divided on textual and graphical languages [KKP+14].
Development of DSL includes several steps: decision that the DSL is necessary, analysis of the domain, language design and implementation [MHS05].
In the first step the aim of the language and its uses should be defined. Definition of the language uses impacts on the concepts of DSL. Common uses of the language are: documentation of knowledge, code generation, test definition and generation, automatic analysis, formal verification, and simulation [KKP+14].
The second step is an analysis of the domain. The analysis includes an identification of the problem domain and gathering knowledge about it from different sources. The results of the domain analysis are the definitions of the domain scope, domain model that defines terminology that are related to the domain and description of domain concepts. The defined domain model is used as an input for the DSL design. The following analysis patterns can be used: informal, formal, and derivation from code. Informal methodology allows to analyze the domain in an informal way. Formal analysis requires application of the domain analysis methods (Family Oriented Abstraction, Specification, and Translation, Ontology-based Domain engineering, etc.). The derivation from code defines domain related terminology by analyzing source code and deployment scripts [MHS05].
The design of DSL can be done by creating DSL from scratch or by developing DSL on the top of the existing language. The latter approach is easier, because reusing of the GPL syntax is available. A DSL designer should not create own type system. Language design can be done

using formal and informal approaches. Informal approach describes the language specification using natural language, including DSL examples. Formal approach requires specification written using one of the semantic definition methods [MHS05].

1.2.3 Ontology based domain analysis

Ontologies describe "explicit formal specifications of the terms in the domain and relations among them" [NM]. An ontology consists of classes, that describe the concepts of the domain, class properties and restrictions. Classes can have subclasses that inherit properties of the parent class. The ontology based analysis consists of the several steps [NM].

Step 1: Determination of the domain and the scope of the ontology. This step includes the definition of the domain and a purpose of ontology creation. It also includes a creation of the questions that will be used for the verification of the ontology completeness.

Step 2: Existing ontologies reusing. It allows to use the already existed knowledge about the domain. There are several ontology databases: Ontolingua, DARPA Agent Markup Language (DAML), RosettaNet, Directory Mozilla (DMOZ).

Step 3: Listing of the important domain-related terms. It simplifies development of a class hierarchy. The definition of the classes and their properties can be chosen from the predefined list of all domain-related terms.

Step 4: Development of the ontology class hierarchy. It includes a creation of hierarchical structure by defining classes and subclasses from the list of terms. There are several guidelines for the class hierarchy definitions [NM]:

- Class hierarchy correctness: The guideline includes the rules for verification of "is-a" relations, transitivity of class relationships, class uniqueness (different classes should not represent the same concept), prevention of Class cycles (two classes should not be the superclasses of each other).
- Class hierarchy sibling analysis: The guideline defines that "all the sibling classes in the hierarchy must be at the same level of granularity" [NM]. It also defines the number of subclasses that the class should contain. The class should not contain only one subclass or dozen subclasses. Such hierarchy could indicate to a modeling problem or to incompleteness of the ontology.
- Multiple inheritance: the guideline allows one class to inherit properties from more than one superclass.
- New class introduction guideline: Set of rules that helps to introduce new classes in class hierarchy: "Subclasses of a class usually (1) have additional properties that the superclass does not have, or (2) restrictions different from those of the superclass, or (3) participate in different relationships than the superclass" [NM]. Another rule defines that the "classes in terminological hierarchies do not have to introduce new properties" [NM]. There are also rules that help to define whether the term represents the new class

or can be defined as a property of the already existed class. If the such property becomes restrictions for properties in other classes, then the new class should be introduced.

- Limiting the scope: the guideline says that the ontology should not include all possible information about domain and properties, but should only include the information important for specific application.
- Create disjoint subclasses: different classes should not contain the common instance.

Step 5: Definition of the class properties (slots). This step adds the additional information to the classes that help to answer the questions from step 1. There are different types of properties: "intrinsic", "extrinsic", parts, and relationships properties. Class properties describe variabilities between components.

Step 6: Declaration of the properties boundaries. It describes allowed values and a number of values (single, multiple, minimum and maximum number of values) that the properties can have. [NM] defines the following attribute value types:

- The string represents the simple text value.
- The number describes numeric values and could be defined as more specific Integer, float or double value.
- Boolean could have two instances: true or false.
- Enumerated represents the list of predefined values of a specific type.
- Instance allows to define the relationship between instances, and should include the list of allowed classes.

Step 7: Instances of the classes creation. The final step includes selecting and creating the instance of the class, and filling it's properties.

1.2.4 Topology and orchestration specification for cloud applications

The Topology and Orchestration Specification for Cloud Applications (TOSCA) standard is a language for the description of a service topology and management procedures for the service creation and modification. A service template defines topology and orchestration that enable the interoperable deployment of the cloud services.

The figure 1.1 shows the main components of the TOSCA service template. A topology model defines a structure of a service. It consists of node templates and relationship templates. These components represent a directed graph.

The node template is an instance of the node type that describes the properties and operations of the service component. Node types define a structure of properties by providing property definition. The actual property values should be filled in node template. The node type can derive properties from another node type. Derivation rules say that XML element of node type properties should extend XML element of the node from which a target node is derived. Requirement, capabilities, instance states and interfaces of the node type can also be inherited.

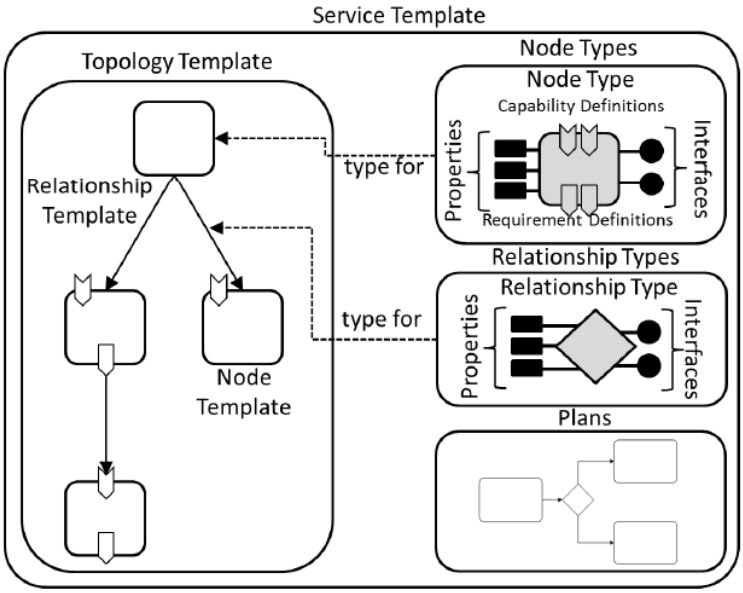

Figure 1.1: TOSCA service template scheme [Tos13b]

Relationship template is an instance of the relationship type that represent a connection between different components and defines the direction by providing source and target elements. Plans describe the management aspects of service instances. Business Process Model and Notation (BPMN) and Business Process Execution Language (BPEL) as well as any language for process model definition can be used for the orchestration. The plan contains tasks that refer to the node and relationship interfaces. It is also possible to call the external services [Tos13b].
TOSCA supports the following use cases [Tos13b]:

- Service as a marketable entities: service developer can create different templates and publish them in the service provider's catalog. Cloud provider can map the published topology on his infrastructure.
- Portability of service templates.
- Service composition allows to compose the service from the components provided by different developers.
- Relation to virtual images: if the cloud provider hosts the services based on middleware stacks (the virtual images), then node in the service template can correspond to such virtual image.

TOSCA supports declaration of requirements and capabilities for the service components. It allows to define dependencies between nodes. For example, a database requires installed Operating System (OS) before the database can be installed. In TOSCA it could be defined by assigning "OS requirement" and "OS capability" to the database and OS nodes. Different requirements of one node can be satisfied by capabilities of different nodes. Declaration

of dependencies could be done by defining requirements and capabilities definitions, which represent instances of corresponding requirement and capabilities types. These types support derivation. Derivation rules say that XML element of requirement or capability type should extend XML element of the corresponding type from which it is derived. Requirements and capabilities could be matched in two ways: using direct relationships between components in the same template, or the matching could be done by the TOSCA container during instantiation.
One node template could be substituted by another service template if they have the same boundaries (properties, capabilities, etc.).
Nonfunctional characteristics (e.g. quality-of-services) are provided by the definition of policies. One node template can be associated with different policies. Declaration of the policies is done by defining policy templates that are the instances of the policy types. The properties of one policy type could be derived from another policy type.
TOSCA primer [Tos13a] defines a processing environment to process TOSCA definitions that are operated by the cloud provider. It consists of the following components:

- TOSCA modeling allows to model service templates.
- Cloud Service Archive (CSAR): an archive file with predefined structure that contains all definitions, artifacts, plans, images.
- TOSCA Container processes the CSAR file and supports all steps needed for provisioning and management of cloud application
- A process engine is an optional component and is used in the case when the service template provides plans.

2 Related work

The chapter gives a review of the work that was done in the analysis of ECM domain.

Functional ECM Framework. Knut R. Grahlmann et. al. review the enterprise content management from the functional perspective. Their work consists of two parts. In the first part the definition of ECM is given, and in the second the Functional Enterprise Content Management Framework (FEF), which describes the functionalities of ECM systems (ECMS), is introduced. The work is based on the reviewing of academic literature about ECM. The found papers were categorized according to four perspectives [GHH+06]:

- A content perspective that includes three views: user view, content view, and system view
- A technology perspective describes "basic technologies used for ECMS" [GHH+06]
- An enterprise perspective "considers organizational, social, and business issues" [GHH+06]
- A process perspective includes "research about both the development and deployment of ECMS" [GHH+06]

Knut R. Grahlmann et. al. found that all ECM definitions from the papers they reviewed could be split in two groups: the first group defines ECM from content and technical perspective, and the second group defines ECM based on enterprise or process perspective. Based on these papers, the authors give the following definition of ECM: "Enterprise Content Management comprises the strategies, processes, methods, systems, and technologies that are necessary for capturing, creating, managing, using, publishing, storing, preserving, and disposing content within and between organizations." [GHH+06]
The second part of their work is dedicated to the description of the functional ECM framework. The authors derive several scenarios, when FEF can be used:

- A description of ECM functionalities in organizations
- A future research
- A comparison of functionalities of the existing ECMS

The authors indicate five guidelines for the framework. FEF should be comprehensible and usable, complete, generic, distinguishing enough, and future-proof. The final framework is shown on figure 2.1

The model consists of four layers:

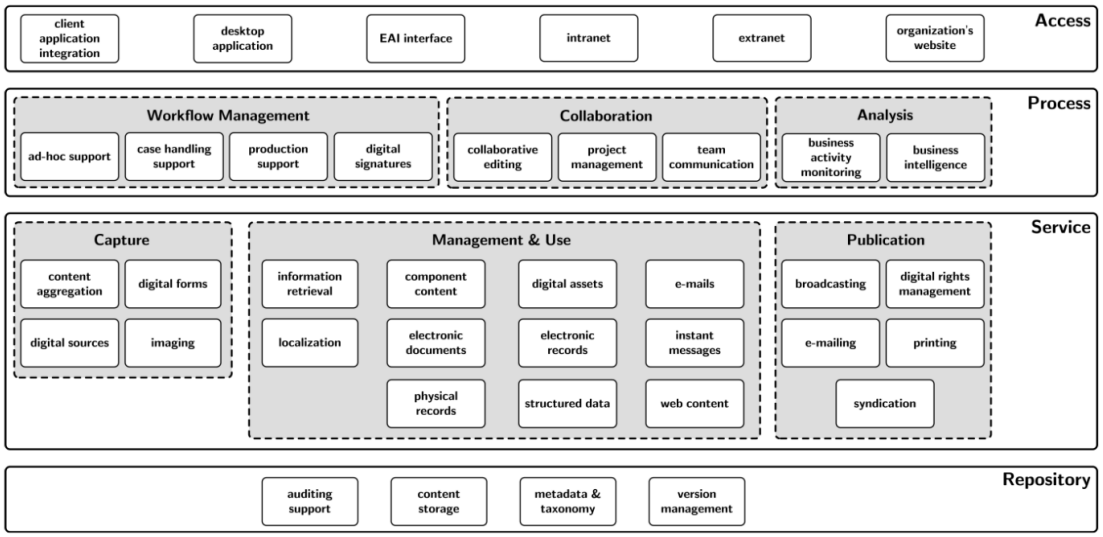

Figure 2.1: Functional ECM framework, [GHH+06]

- "Access" layer includes functionalities for users and other systems for interacting with ECMS.
- "Process" layer describes functionalities for workflow management, collaboration and analysis.
- "Service" layer provides capturing, management and publication functionalities of content.
- "Repository" layer includes functionalities for storage and preservation of content.

The evaluation of the FEF was made by examining ECMS in three organizations and reviewing the framework by ECM consultant. Examination of ECMS in particular organizations showed that all these companies use software components from different developers. The FEF helped to describe, which functionalities different software products support. The ECM expert claimed that "the FEF is in line with experience from ECM-projects at various clients" and can be "easily mutable and considered future-proof" [GHH+06].

Enterprise content management: an integrated perspective on information management. Tero Päivärinta et. al. introduced the ECM framework from the organizational point of view. Their work is based on the analysis of 56 ECM projects shared by the AIIM organization [PeM05]. The developed ECM framework is shown on Figure 2.2 and consists of the following components:

- The concept of a content model refers to content structure and view, its life cycle, metadata, and taxonomy. It allows to understand, how the content should be transported to the ECM system;

- Enterprise model defines the business workloads and roles in the organization;
- Infrastructure describes the technological aspects of ECMS implementation such as standard application and tools integration throughout content lifecycle, user interface development to content management, software, hardware, and technology updates, implementation of security issues;
- Administration of ECM refers to policies, regulations, and administrative procedures to content management.
- Change management indicates to the changes between enterprise and content models over time.

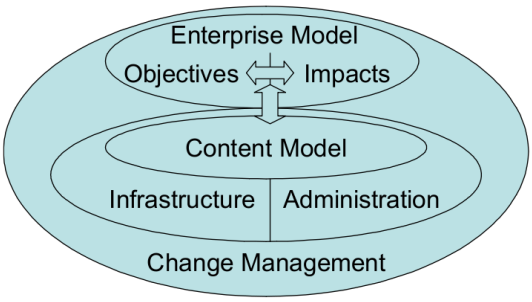

Figure 2.2: Major ECM issues, [PeM05]

The authors indicate the following ECM objectives: improved collaboration, reliability and quality of information content, tracking of transactions, cost savings, compliance to standards and legislations.

California's department of technology definition of ECM reference architecture California department of technology defines the ECM reference architecture based on the AIIM's ECM 101 poster and derives five key areas of ECM [oT14]:

- "Capture" includes different technologies for document creation and capturing, and formats, in which the content can be saved.
- "Manage" defines the operations related to content lifecycle management and includes the following components: document management, web content management, forms management, records management, digital assets management, business process management and workflow, and content management services.
- "Store" is responsible for persisting and accessing data.
- "Preserve" is responsible for long-storage persistence of data.
- "Deliver" returns content to the user and includes formats and technologies for content delivery.

The ECM reference architecture is based on these key areas and consists of three vertical layers (Figure 2.3):

- Capture and deliver components, which include the technologies for creating/capturing and delivering the content.
- Manage components include components and core services for management of the content lifecycle.
- Content repositories include technologies for storage of different type of content.

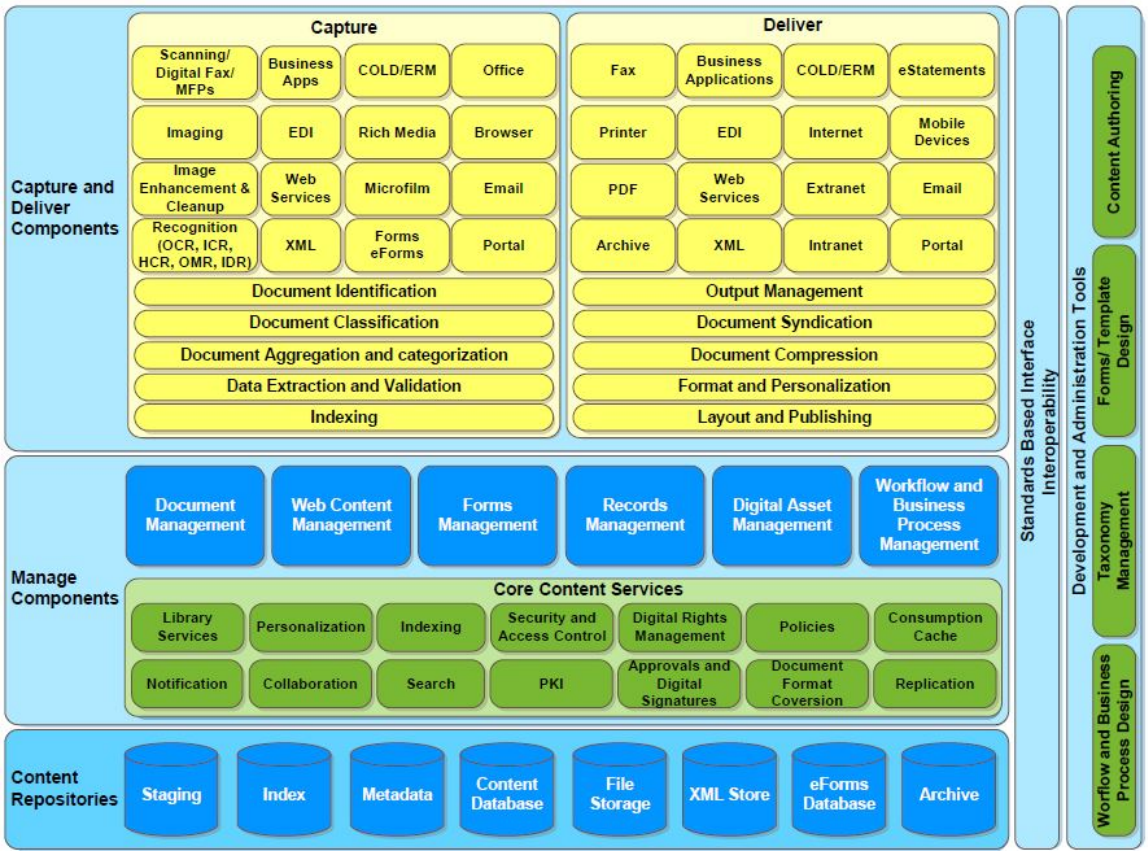

Figure 2.3: ECM reference architecture, conceptual view, [oT14]

The horizontal components of the reference architecture are applied to all vertical layers and includes interface interoperability standards and development and administration tools. The document focuses on ECM concepts at the enterprise architecture level and does not include other areas for implementation of ECM solutions like legislations and corporate policies. It also defines, which components should be considered during ECM implementation and does not define, which capabilities should be provided.
The described frameworks were used for the classification of the concrete ECM solutions and for the definition of the ontology classes.

3 Domain specific languages development methodology

DSL is the language that is bounded by the specific problem domain. It consists of semantics and syntax. Semantics define terms and their meaning that describe the problem domain. The terms and their definitions form DSL vocabulary. Syntax declares rules that define how terms are related to each other. Declaration of the DSL semantics was done by definition of the ECM ontology. DSL syntax is declared by TOSCA language.
Development of DSL includes several steps: decision that the language is necessary, analysis of the domain, language design and implementation. The detailed description of the DSL development steps is done in section 2.2.2 of the introduction chapter.
This chapter starts with an analysis of commonalities and variabilities of the ECM domain. Then the definition of the domain scope and the description of the applied methodologies for the domain analysis and DSL design using TOSCA is introduced. The result of the domain analysis, the abstract description of DSL and the concrete implementation using TOSCA language is done in chapter 4 "DSL Modeling results".

3.1 Commonality and variability analysis of ECM solutions

For the definition of the commonalities and variabilities of ECM solutions I defined the components that are provided by IBM, Alfresco, ECM. These companies were selected from the leader group of Gartner's magic quadrant [GSC+14]. Then I arranged the components according to the FEF [GHH+06]. After that I defined common operations for each component type.
I started the definition of the commonalities and variabilities with the analysis of the components that form foundation of the ECM system.
Alfresco ECM foundation consists of three layers:

- Content repository describes low level IT components that are used for the preservation of content and its metadata. It consists of file system and relational database.
- The application server unites content, control and collaboration services. The content services allow to manage content lifecycle, convert content from one type to another, extract metadata and generate tags. Control services enable content workflow that consists of a sequence of steps. Collaboration services consist of a social graph that represents people and relationships between them, continuous personalized feed, creation and edition of interlinked webpages, threaded conversations.

- The client layer describes the services that allow user to access the content stored in Alfresco repository. It includes web, mobile and office clients.

The architecture of the EMC ECM foundation solution "Documentum" consists of the following groups [EMC08]:

- The kernel group consists of the security (audit, access control, authentication, etc.) and the repository services (lifecycle, storage, users, etc.), and repository infrastructure (database, full-text index, etc.).
- The application services group consists of the process (analytics, project tracking, etc.), content (transformations, classification, etc.) and compliance (records, retention, etc.) services.
- The experience group defines an access layer of ECM domain that allows users to interact with an ECM application and kernel service and includes WDK based platform for deployment extensional applications. The interaction between experience layer and application services is implemented through the Application Programming Interface (API).
- Tools allow design, configure and administrate ECM solution.

IBM P8 ECM foundation consists of three main layers [ZBO+11]:

- Information infrastructure unites data and storage services, that allows to call the data from data stores and provides abstraction between P8 platform and the data and storage services.
- ECM/BPM services are defined by the content and process engines and enable the core content and process operations for content management.
- Input, presentation and output services allows to interact users with ECM solution. They also provide API for third party applications. According to the FEF that was described in Chapter 3 "Related work", all described ECM solutions can be grouped into the following layers: repository layer, service and process layer, and access layer.

The examination of the ECM vendors architectures showed that all ECM related terms can be grouped into the following layers according to FEF [GHH+06] (Figure 3.1):

- Access layer is defined by capture and delivery terms and includes the technologies and components for the content creation, acquisition as well as for showing and exporting the queried content (APIs, Web Clients, etc.).
- Manage Layer includes service and process layer and defines services for the management of enterprise content. According to the examination of technical documentation, management layer consists of the core components that describe core ECM services, and extension components that add functionality to the core component (e.g. Records Management).
- Repository Layer includes technologies and components for the content storage (Databases, File Storages, etc.).

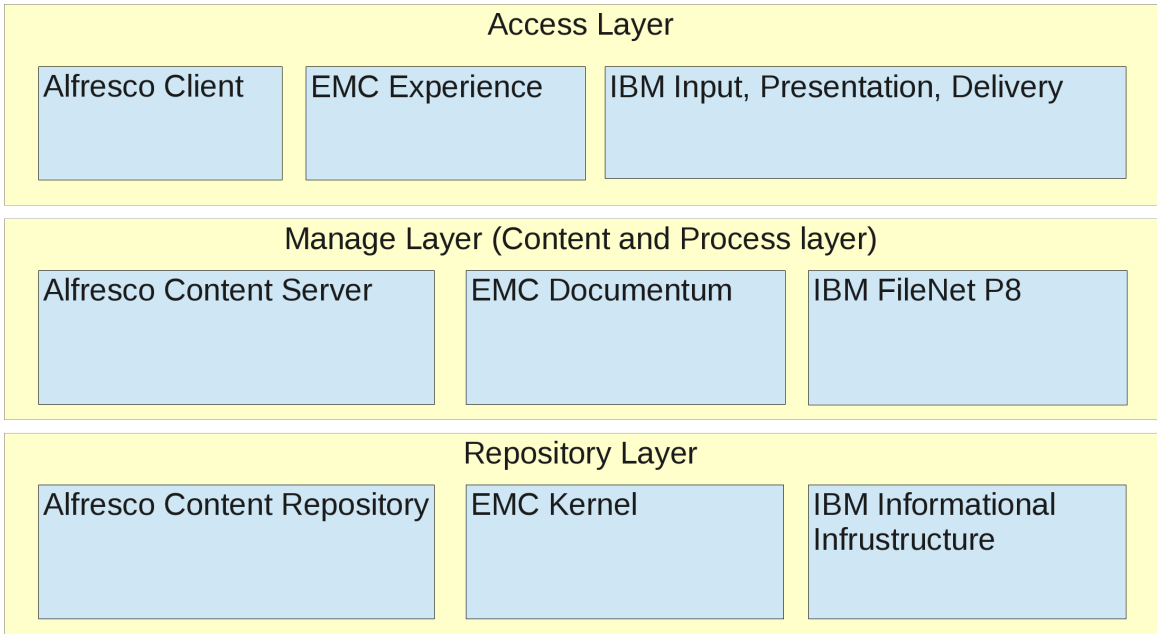

Figure 3.1: ECM layers

The analysis showed that most of the solutions have the same general architecture: there are core components, that form the foundation for ECM solutions. All examined vendors have the repository and library services as the central component, which provide core functionality for capturing, storing, retrieving, and management of the content. There are different definitions of ECMS core components. Alfresco defines the central component as an content application server. IBM's P8 core services unite object stores - the storage where content and its metadata are stored, content engine - services that provide core content management functions, and process engine - core component, that provide basic business process management operations [ZBO+11]. Academic literature [GHH+06] distinguishes "Repository", that provide storage functions for content and metadata, and library services, which implement access interfaces to that data. In my model I used definition "Repository", which includes storage for content and its metadata, and the library services that implement core Create Retrieve Update Delete Search (CRUDS) operations. This repository stores the content and information about it in the storage (file system, database, etc.), and manages access to the repository using Directory Services (DS) Server (DS server could be implemented as a part of ECMS solution, or the customer's DS server can be used). All additional components (Records Management, e-discovery, advanced search services, etc.) extends the functionality of the repository. Third party software could access repository through the API. Most of the ECMS developers uses Content Management Interoperability Services (CMIS) standard to provide an interface to the repository. This standard describes the main functions that should be supported by the repository. The Figure 3.2 shows the generic architecture of the ECMS system.

After the analysis of the different architectures, I examined the particular components that are provided by each company. I started with the core component that forms a foundation

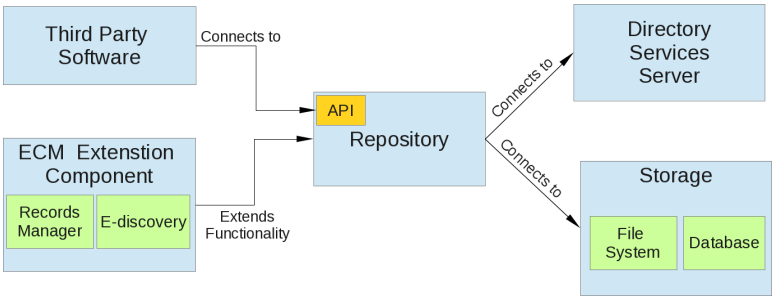

Figure 3.2: Generalized architecture of ECMS

of ECM management solution. The following components were examined: Alfresco Content Repository, ECM Documentum, IBM FileNet P8. For each component were defined workloads that consist of operations that are supported by the component. The common workloads were defined. A list of the ECM component and supported workloads is shown in Table 3.1. The analysis of the supported workloads shows that examined ECM solutions have common

ECM Component	**Supported workloads**
Alfresco Content Server	Folder services, versioning services, check-in, check-out, audit services, Authority services, Permission Services (ACLs), Content lifecycle management, transformation services, metadata extraction, tagging, workflow support (jPDL process language for definition states, transitions, tasks, events, actions)
ECM Documentum	Check-in, Check-out, versioning, Full-text indexing, Security (authority, permission control (repository level, ACLs)), accountability (auditing), Process Management features (document workflows and automated lifecycle stages)
IBM FileNet P8	Check-in, Check-out, versioning, security (authority, permission control),,lifecycle management, foldering, full-text indexing, workflow supports (step processes and user inbox operations)

Table 3.1: List of supported workloads by different ECM vendors

workload names. The more detailed analysis showed that the workloads with the same names can include different operations that the customer can execute. For example, all examined components support workflow management, that describes document worklflow operations. EMC Documentum workflows management workload includes also operations for automation of lifecycle stages. The same analysis was done with other ECM components (records management, e-discovery, etc.) and the workloads with the same names and different supported operations were found.
The analysis of commonalities and variabilities of ECM solutions from different vendors showed that the components from the examined vendors can be classified according FEF [GHH+06]. Additionally all examined solutions have the common generalized architecture:

there is core a component that consists of the repository and library services. All other components extends the functionality of that component. The variabilities are provided by the workloads that are associated with the ECM components. The workloads with the same name can provide different operations. These variabilities should be solved by ECM DSL.

3.2 ECM domain analysis

Most of the resources dedicated to the DSL design does not cover in detail methods for the domain analysis. In this section the overview of the domain analysis methods will be given, and an application of the ontology based methodology to the ECM domain analysis will be described.
The goal of the domain analysis is to define terms of the ECM domain, their commonalities and variabilities, and interconnections between them. The result of the analysis is the domain model that will be the input for the DSL design. The abstraction, information hidden and prediction of future changes are important terms in the domain analysis.
There are different patterns for the domain analysis: informal, formal and derivation from a code [MHS05]. The formal facility includes different methodologies like family-oriented abstraction, specification, and translation (FAST), ontology-based domain analysis, and others. During the studying different methodologies, I decided to use the ontology based methodology, that describes "explicit formal specifications of the terms in the domain and relations among them" [NM]. The ontology defines classes and their hierarchy. According to the ECM domain, classes define terms that form the DSL vocabulary. The ontology based methodology has relative concepts as a TOSCA metamodel (definition of classes, property types, relationships) and could be mapped to TOSCA terminology. In the following sections application of the ontology development steps defined by [NM] will be described.

3.2.1 Determination of the domain and scope of the ontology

The development of the ontology model starts from the definition of the domain and its scope, and should cover the following aspects: description of the domain, purposes for the ontology creation, definition of the main users, who will use the ontology, and the list of the questions for approval of the domain completeness [NM].
First, the definition of ECM will be introduced. AIIM defines ECM as "the strategies, methods and tools used to capture, manage, store, preserve, and deliver content and documents related to organizational processes"[AII14]. Gartner divides the term ECM in two frameworks: a strategic framework and a technical architecture. The strategic framework allows companies to increase the productivity, take control of their content, enable content-centric processes. The technical architecture consists of the platform - set of services that provide core ECM functionality, and additional modules that extend platform's functionality [GSC+14]. Both these definitions emphasize that the successful implementation of ECM includes not only the usage of a technical system that allows to manage of informational assets, but also an appliance of strategies for effective content management. Based on these definitions I focused the

ECM domain analysis on the content lifecycle and terms definition of ECM. Term definition helps to understand the terms and their workload that form an intelligible to business customer language, and includes the analysis of the academic literature, technical documentation of particular ECM solutions, and the discussions with the developers of ECM systems. The content lifecycle view allows to understand factors that impact on enterprise content during its life cycle. It helps to describe the existing enterprise architecture related to content acquisition, storage, management, and delivery. This view includes the analysis of standards, corporate governance, and academic literature.
The purpose of the ontology creation is to identify the functional and non-functional requirements to the ECMS. These requirements will be used as the inputs to the ECM DSL.
The primary users of this ontology are business customers, who want to implement ECM solution in their organizations, ECM architects, who will design ECM solution topology, and cloud providers, who will deploy ECM system. Business users have to define functional and non-functional requirements to the ECM solution . These requirements can be declared as a "request for proposal" document, where customer defines requirements using terms that he/she understands. Architects have to design ECM service templates according to the customer requirements. Providers have to transform these requirements of particular ECM architecture.
A questionnaire will be used to check whether the domain model is complete. The following questions were stated:

- Is the presented language useful for the definition of the quantitative and qualitative characteristics and policies of ECM System?
- Are ECM components described using high level abstraction and are they independent from particular components from the different vendors (IBM, Alfresco, Oracle, etc.)?
- Does the presented ECM template allow to define components that should be deployed?
- Is it possible to define Quantitative characteristics (max number of concurrent users, average size of files, number of file shares, etc.) from the provided ECM service template?

[NM] defines the guidelines for ontology creation. One of these guidelines tells that the scope of the Ontology should be limited by its application. Because of that the developing ontology will be limited by the definition of the common classes that are important in ECM.

3.2.2 Definition of classes and class hierarchy

The definition of a class hierarchy starts from the searching of the already existed ontologies that could be reused. Then the list of terms will be created for classes and their hierarchy definition. In the last step the properties of the classes should be described.
For the determination of ECM terms I used the following sources: ontology databases, academic literature, technical documentation about ECM, developer's documentation, deployment scripts provided by IBM. I used the following searching constructions for the ECM terms: "ECM", "Enterprise Content Management", "Content Management", "ECM Reference architecture", "ECM blueprint". Reusing of the existing ontologies allows to develop a

domain model based on the already existed knowledge. There are several ontology databases: Ontologia library, DAML, RosettaNet, DMOZ.
Examination of the ontology databases gave no result. Studying of the academic articles showed that some work was done in defining ECM architecture from different points of view. [GHH+06] introduces the Functional ECM Framework, which describes the different layers and components of ECMS. [oT14] also overviews ECM from functional perspective but has a different definition of layers and components. It adds two additional vertical layers: "standard based interface interoperability" and "Development and Administration tools" These two functional frameworks were used as a basis for definition of functional requirements of ECMS.
[PeM05] describes ECM domain from the Enterprise Perspective. The connection between enterprise architecture and a content model is shown. This framework shows that the company should define not only requirements to the ECMS, but should also provide information about the enterprise architecture.
Studying Academic literature dedicated to Information Lifecycle Governance (ILG) showed that there are different factors that impacts on ECM. There are different standards, government legislations and regulations provided by governments that every industry has to comply. Companies develop a corporate governance, which includes best practices, policies and procedures to follow the compliances. In addition to the business requirements, there are also legal requirements that should be mentioned during ECM implementation [Sma14]. These requirements include:

- Records management complies enterprises to store the documents that have legislation value as records. Records can not be modified and should be stored and disposed according to the records retention schedule.
- Legal holds complies enterprises to hold records in the case of a government investigation. These records shouldn't be disposed according to the retention schedule during the investigation
- E-discovery allows companies to search the records and content according to the investigation cases.

IBM's questionnaire allows to define terms of ECM and provides additional knowledge about information that should be gathered from the customer. This information includes qualitative and quantitative requirements of ECM. The quantitative characteristics include: average size of the different type of content, maximum upload/search/retrieve operations per time unit (week/month/year), peak usage periods, number of e-Discovery tasks, number of File Shares, number of ECM systems, maximum and average number of concurrent users. The qualitative characteristics include a description of basic archive operations such as Ingest, Classification, Indexing, Search, Retrieve, Update, eDiscovery, Retention or Records Management. This information helps to the cloud provider to define which components should be deployed and to build the IT architecture that will fulfill Service Level Agreement (SLA).
The technical documentation provided the information about the characteristics of the ECM components. For example, IBM has a "Content Collection" component, that allows to acquire the content from enterprise resources (file and mail servers, other repositories, etc.) and store it in ECMS. For the correct deployment of this component, the information about the content

sources should be provided by the business customer. For example, the customer could have the requirement to store all content from a mail server in the ECM system. For these purposes he/she should provide the information about the characteristics of the mail services such as type of the server (Microsoft Exchange, Simple Mail Transfer Protocol (SMTP), Internet Message Access Protocol (IMAP), etc.), location of the server, IP address of the server, etc.
Additionally, deployment scripts provided by IBM have been studied. They provided information about the parameters such as users, groups, roles, and others. They also define the administration operations that each ECM node can support (install, uninstall, start, stop, etc.).
The result of the literature analysis is the list of terms that can be used for the class and properties definition, The list is shown in Table 3.2.

Literature Source	**Term Type**	**Terms**
Functional ECM Framework [GHH+06]	ECM Layers	Access Layer, Process Layer, Service Layer, Repositories
California's ECM Reference Architecture	Capture Components	API, Document Scanning, Digital Faxes, Recognition Technologies, Automatic capture, Semi-automatic capture, Image Enhancement and clean-up, Document Identification, Document Classification, Document Categorization, Data Extraction, Indexing, Metadata Creation, Content based indexing, Metadata based indexing
	Deliver Components	On-Premise ECM, Software-as-a-Service ECM, Hybrid ECM, Transformation Technologies, Converters, Viewers, Compression, Syndication, Security Technologies, E-Signature, Public Key infrastructure, Distribution and Output Management, Internet, E-mail. Fax, Electronic Data Interchange (EDI), Extensible Markup Language (XML), Mobile, Multimedia, Paper
	ECM Management Components	Content Management Services, Document Management, Web Content Management, Forms Management, Records Management, Digital Asset Management, Core Management Services, Library Services, Check-in, Check-out, Indexing, Search Services, Digital Right Management, Security and access control, E-Signature
	Store Components	File System, Database, Storage Technologies, Storage Area Network (SAN), Network-attached storage (NAS)

Continued on next page

Literature Source	Term Type	Terms
Enterprise Content Management: an Integrated Perspective on Information management	Layer	Content model, Enterprise Model, process-based Enterprise model, role-based Enterprise Model, project-Based Enterprise Model, Infrastructure
IBM Questionnaire	Qualitative Characteristics	Ingest, Classification, Indexing, Search, Retrieve, Update, Retention, Records Management, eDiscovery, Holds, Reporting, Auditing, Disposal, Exporting, Compliance workflows, External Services
	Quantitative Characteristics	Maximum number of concurrent users users, Average Number if concurrent users, Number of Docs loaded per time, Number of Retrieves per time, Number of Updates per Time, Number of FileShares
	Ingest Characteristics	Import methods, Windows desktop via Navigator, File Servers, Protocols (Secure File Transfer Protocol (SFTP), Short Interframe Space (SIFS)), Batch import via IBM Content Collector, Batch import via CMIS, Batch import via Web Services (WS), Batch Import via API, Number of Filers, Number of File Shares, Unstructured Repositories, Number of documents loaded per time, Average document size, Document Formats (Word2k3, PDF, TIFF, JPEG), Black list of document types, Steaming media, Archive Storage, Pre-processing Requierements, Electronic Report Distribution, E-mail Notification
	OnDemand Characteristics	High speed front-end data loading solution, PDF Indexer, Report Mining, electronic Report Distribution
	Scanning Characteristics	Partner Solution, Number of documents per day, OS Client
	Viewing for documents by Client Application	Content Navigator, Native Viewing, Plugin or application, Applet, electronic Discovery Manager (eDM), electronic Discovery Agent (eDA), Rendition, Rendition formats, Native, PDF, HTML, XML, EDRM

Continued on next page

Literature Source	Term Type	Terms
	Parametric Search Characteristics	Taxonomy, Document Classes, Search Attributes, Relationships, wild card search, sorting, Retention Management
	Full text search characteristics	document types to be indexed, volume of full text indexed documents per day
	Records Management Characteristics	Retention Management, File Plan, RM Standards (Base, DoD, European), Integration with viewers
	Content Classification	Automatic classification, classifier, statistical analysis
	Security	Lightweight Directory Access Protocol (LDAP), LDAP Version, LDAP Type (Domino, AD), Number of domains, Secure Sockets Layer (SSL), Single sign-on (SSO)
	Internationalization	Languages
	Business Continuity	Backup, Restore, High Availability, Disaster Recovery
	Resources and Responsibilities characteristics	Departments, Roles
	Communication and Networking	Connection to Archive, Document Metrics, Network Bandwidth, Host Naming Conventions, IP Adresses
Enterprise Content Management: A business and Technical Guide [Cam11]	Acquisition	Corporate desktops, INTERNET, e-forms, scanning centers, multifunctional devices, repository, e-mail, enterprise agreement formats, original formats (paper, electronic), cataloging, indexing.
	Storage	Enterprise storage (distributed, federated, cloud), Enterprise repository (shelves, file storage, etc.), Number of versions, access control, digital right protection, retention life of content.
	Delivery	Searching, web page, downloading, printing.

Table 3.2: List of the ECM domain terms

After the gathering information about the ECM terms, the classes and class hierarchy could be defined. There are different approaches for class hierarchy modeling:

- Top-down approach: the class definition starts from the more generalized classes, and then more concrete classes are defined

- Bottom-up approach: the definition of classes starts from the initialization of the particular terms, which then are combined with more general classes
- Combination development: combines top-down and bottom-up approaches. The definition of hierarchy could start from the concrete terms and then more specialized and generalized classes can be defined.

I used the combination development approach. The method described in [NM] for definition of the class hierarchy was applied: first, the classes from the term list were selected, then the sub- and parent- classes were defined. The next step was to define the properties of the classes, which were derived also from the term list. The resulted classes were examined, and the more generalized and specialized classes were defined. The generalized class was introduced if several classes contain the same parameters and are part of the same parent class. During the definition of the classes the guidelines for class hierarchy correctness, sibling analysis, new class introduction and scope limitation described in [NM] were used. The more detailed definition of these guidelines is described in section 2.2.3 of the fundamentals section of the introductory chapter.
The common layers and definition of generalized architecture that were described in commonality and variability analysis section were used as a starting point for the class hierarchies definition. Then more concrete and more generic classes were selected.
Additionally, IBM Questionnaire defines the qualitative and quantitative information that should be provided by the business customer about already existed enterprise infrastructure. That information is used by ECMS providers during deployment stage. Quantitative data helps to choose correct Information Technology (IT) architecture that will satisfy Service Level Agreement (SLA). Qualitative data allows the provider to choose the correct components. For example, the core ECM component requires the information about LDAP servers for security considerations, or content collector requires the information about the enterprise storages (servers, unstructured repositories) where the data that should be collected is stored and the corporate taxonomy that defines in which way a content should be categorized.

Definition of class properties and it's boundaries. The property definition usually is done as a part of the class hierarchy definition. Properties should have two important boundaries: cardinality and value-type. Cardinality defines the minimum and maximum number of the parameter occurrence. For example, the parameter "Standard Browsers" of the policy class defines the list of the browsers that should be supported by Web Client. This parameter could contain several values of the same type ("Google Chrome", "Mozilla Firefox", etc.). If the minimum number of values equals 0, then it means, that the parameter is not required. Value-type defines the type of the value that the property could contain and could be one of the following values: string, numeric, boolean, enumerated, instance. More detailed description of value-types is done in the section 2.2.3 of introduction.
The definition of the parameters was done by the selecting of the remaining terms from the list (Table 3.2). IBM questionnaire was used as a basis for properties definition. Not all the terms from Table 3.2 were used for declaration of classes and their. Only terms that allow to create the ontology that will describe the core principals of ECM and enable a creation of the extensible class hierarchy were selected. The qualitative characteristics of IBM's questionnaire,

the terms defined in functional ECM framework, California's ECM reference architecture, and integrative perspective on information management were used for definition of classes and their hierarchies. The properties of classes were defined based on the quantitative parameters of IBM's questionnaire.
The result ontology of the ECM domain is described in "DSL modeling Results" chapter.

3.3 DSL design methodology

DSL defines semantics (terms and their definition that form a language vocabulary) and syntax (rules for logic connection of vocabulary items) that describe the problem domain. There are two main types of domain specific languages: internal and external DSLs. The internal DSL is based on the existing GPL, and the external DSL is developed from the scratch.
The internal DSL based on the TOSCA language was used for definition of the ECM workloads. Exploitation of TOSCA allows to develop the language based on the already existed syntax. TOSCA is the standard that was developed for cloud services deployment automation. This section is focused on the description of the TOSCA application for ECM purposes and definition of basic features that should be supported by the language. The detailed description of the TOSCA standard was done in the introduction chapter.
TOSCA was selected as the basis of DSL because it was designed for supporting portability of topology templates and services composition. Portability means that the service developer could define topology, plan and policies and publish them as a service template in the catalog. Cloud provider could map this service template to his infrastructure. A service composition allows the cloud provider to use the topology templates from different developers [Tos13b]. Applying to the ECM domain, it allows to define the high level architecture of ECMS and publish it in the catalog. Then, cloud providers could map this architecture to their infrastructure and substitute high level nodes with the exact topologies provided by ECM developers. TOSCA also describes the processing environment for the service templates [Tos13a]. It allows cloud providers to automate the provisioning of the ECMS and reduce the total time needed for the solution deployment.
The usage of the TOSCA will be limited by the usage of the nodes, relationships, requirements and capabilities, and policies. These components allow to model the syntax of the language. Instantiation of the TOSCA types is done in the service template. It defines the boundaries, topology of the components, policies, and plans.

Nodes definition The classes of the ECM ontology represent the semantics of the language. It describes ECM terms and the relationships between them. Applying to TOSCA, ontology classes could be mapped to the node types, and the properties that describe the links to other ontology classes could be mapped to the relationship types. The node types have hierarchical structure that allows to use almost the same hierarchy as in the ECM ontology.
Instead of the ontology classes, TOSCA node types do not support multi inheritance. It means that the classes that have more than one parent should be declared in a different way. During domain analysis, an assumption was made that some nodes that could be defined in future could be inherited from multiple classes. For example, an enterprise portal, that could be

defined in future by the service architect, can derive properties from both capture and deliver nodes, because this term can be used for the description of content uploading and retrieving. On the other hand, "scanner" term could be used only for description of the content capturing. Unification of the classes and definition of the property boundaries could be applied as a solution. Capture and delivery classes could be united in one class named "Access", that will describe the properties for both term types. If the exact node (e.g. scanner) does not support one of the properties (e.g. retrieving formats), then that property could be omitted (minimum number of parameter occurrence should be set to 0).
TOSCA node types support declaration of requirements and capabilities, that can be used by the processing environment for the definition of dependencies between ECM terms. Each requirement and capability definition contains the lower and the upper bound, that the node should match or can serve. For the requirement definition the default lower and upper bound is one. If the lower bound equals zero, then it means that this requirement is optional. The upper and lower bounds for a capabilities definition equals one. In ECM DSL definitions of the requirements and capabilities can be used to validate a completeness of the topology template. For example, "Records Management" node could have "File Plan requirement" that can be satisfied by "File Plan capability" that is provided by "File Plan" node. If the user of the language adds "Records Management" node in the topology, then he/she has to add "File Plan" node that describes Enterprise File Plan.
The TOSCA node types allow to define interfaces and operations. Each interface contains the operations that are supported by the node, and can be used in the plans for defining the sequence of deployment actions. In ECM DSL interface describes actions that are associated with the ECM term. For example, the following operations are related to "ECM Repository" term: "create document", "upload document", "retrieve document", "check-in", "check-out". Interfaces help Cloud provider to map DSL terms to ECMS components and will be defined only for the ECM related terms.
The user interface will contain the list of supported by the node operations (e.g. for "ECM Repository" it could be "check-in", "check-out", "create", "retrieve", "update", etc.). For the business customer such listing allows to define which functionalities are supported by the node. For cloud provider, it gives the ability to select the correct substitution node for the term, that support same operations.

Relationships definition Relationships in TOSCA are represented by relationship types and templates. The relationship type defines the type of one or more relationship template. Ontology properties that have references to another class can be mapped to the relationship types. It is not necessary to create relationships for all ontology references. According to the TOSCA standard [Tos13b] processing environment can create relationships between nodes during the execution process depending on the requirements and the capabilities of the nodes. Relationship types could inherit the properties from the parents. The valid source and the valid target types can be specified by defining the requirement and capability parameters of nodes. Sources and targets define the restriction of relationship types. The relationship could be established between nodes that contains same requirement and capability as the relationship's source and target types.

Policies definition the definition of the policies in the ontology was made by introducing "policy" class that describes corporate policies and quality of service requirements. The TOSCA standard allows to create policy types that represent classes with declared properties that could be applied to the nodes. Policy template allows to instantiate the policy type by providing concrete property values. Several types of policies could be related with one node. For the ECM domain the policy properties represent the non-functional characteristics that could be included in SLA, or could be provided by local litigation (e.g. location of ECM storage that stores personal or confidential data could be restricted by the law). These parameters are included in the policy type and are assigned to the "ECM Node". All other ECM related nodes are derived from this node and inherit the policies. The user of DSL can provide exact values for the properties in topology template. The concrete syntax of ECM DSL using TOSCA is described in section 4.2.2 of "DSL modeling Results" chapter.

4 DSL modeling results

This chapter describes the results of the ECM domain analysis. First, the ECM Ontology was defined. It consists of the terms and their meaning that form the semantics of the language. Section 4.2.1 provides the abstract description of the ECM language that is based on the defined ECM ontology. The description of the language was done in informal way and is based on the example.
Section 4.2.2 provides the definition of the ECM language using TOSCA syntax. DSL that is defined in this section extends TOSCA XML schema.

4.1 ECM domain model

The result of the domain modeling is the ontology that defines ECM classes, properties and relationships between them. The ontology class represents an ECM term. Definitions of the ECM ontology terms are presented in Appendix A. It consists of the definition and description of the classes and the class hierarchy, parameters of the classes and their boundaries. This section describes in detail the ontology structure and it defines a reason, why such structure was selected.
I avoid using the word "component" during the definition of a language vocabulary. It was done to prevent misunderstanding of the language terms. The classes that were defined in the ECM ontology does not describe components of ECMS. They describe the terms that are used by business customers to define ECM requirements. Different properties can be associated with each term. In the thesis I replaced word "component" with the following words: "term", "node" and "instance of a class".
As was described in "Domain specific language methodology" chapter, all components of ECM solutions from different vendors can be classified using following layers: access layer, manage layer, and repository layer. These layers was selected as the starting point for the ontology definition. I defined "ECM" class as a root class for ECM access and manage terms. It allows to inherit properties by all classes that are derived from "ECM" class. The "interface" parameter defines the operations that should be associated with ECM terms. As was shown in chapter 3, workloads with the same names that are defined by different vendors, could declare different operations. The "interface" term should provide the normalized definition of operations that are associated with ECM term.
Repository layer represents low level components that describe how the information is stored in the ECM solution. For this layer I defined "Execution environment" class that describe all terms related to IT infrastructure components of the ECM solution. After that I defined more concrete ECM classes. For the ECM access class the following common terms can be

defined: web client, API, content collection. All examined vendors provide web solutions for the accessing ECM components (Alfresco Web Client, EMC Webtop, IBM Navigator). All analyzed vendors also support access to ECM functions through API. The variabilities of APIs are defined by the types of the interfaces (Java, web services, etc.). For the ECM manage class were created subclasses that define architecture of core ECM components. As was described above, manage components can be divided into core component and extension components. I created "ECM Foundation" class that describes core services that should be presented in all ECM solutions, and "ECM extension" class that describe the terms that add additional functions to the foundation.
Examination of the Record management components of different vendors showed that they requires information about enterprise information structure (information taxonomy, file plan, retention schedule). The "Enterprise" root ontology class was introduced. It includes the subclasses that allow to describe business and IT infrastructure of the enterprise. Examination of the IBM questionnaire showed that the characteristics of the enterprise IT infrastructure should be defined. For these purpose I introduced the "Enterprise IT" class that allows to define software and hardware resource for the content capture, storage and delivery. The integrated perspective on information management introduces definition of the content model [PeM05]. According to this framework I introduced "content" class that allows to define parameters of that should be stored in ECMS. It allows to define different attributes that can be analyzed by the cloud provider (metadata that should be extracted, supported mimetypes, etc.).
A simplified class hierarchy of the main ECM terms is shown on Figure 4.1. Black lines show inheritances between classes. Dashed lines show root classes of the ontology. The relationships between classes are not shown on this figure. According to the class definition guidelines [NM], classes of one layer should have the same level of abstraction. For example, "ECM Access" class and "Records management" class should not be the sibling classes.The ontology guidelines also tell that classes can have multiple parents.
The following sections describe in details the ontology classes.

4.1.1 Classes and properties description

The ontology root layer is defined by the following classes: "Content", "Enterprise", "ECM", "Policy", "Execution Environment" and "Relationship".
"Content" allows to describe the content types of the enterprise information that are acquired and/or created by enterprise departments. Introduction of the "content" class allows to describe information that is captured or created by the enterprise. The properties that are defined for this node are defined in Table 4.1.

Content Class Properties

Name	**Value**	**Minimum occurrences**	**Maximum occurrences**
Department	String	1	1

Continued on next page

Content Class Properties

Name	**Value**	**Minimum occurrences**	**Maximum occurrences**
ContentType	Enumerated: paper OR electronic	1	1
ContentFormat	Enumerated	0	Unbounded
AverageContentSizeInMB	Integer	1	1
TaxonomyCategory	Class DocumentType	1	1

Table 4.1: Properties of the content node class

"Department" defines the name of the enterprise department that acquires the content of the specified type. The boundaries tell that one type of the content could only be captured by one department. "ContentType" specifies the type of the format that is captured by the organization and can be paper or electronic."ContentFormat" describes a format of an electronic content. One Content object could have several formats. "AverageContentSizeInMB" specifies an average size of the document with an associated document category. This value is used to define load to the ECMS. During the execution of the ECM model, Cloud provider can calculate, the average size of all uploaded documents. This information can be used during the design stage of the exact ECM solution to balance the workload and provide appropriate Quality-of-Service. "TaxonomyCategory" defines related taxonomy category. This property allows to define a taxonomy of enterprise information and has dependencies with different ECM terms (e.g. content collector can decide, which documents by specifying appropriate taxonomy category). Information can be categorized by document types, business processes to which it relates, departments, business entities, and others [BB07].
"Enterprise" class has no properties and is introduced to support the logical structure of the ontology. This term describes information and IT structure of the enterprise. "Information structure" defines organizational structure of content, records hierarchy, and others. "IT Structure" defines hardware and software components that are used by enterprises to capture, store and delivery of the content.
"ECM" class is a generic class for the ECM related terms. A declaration of its subclasses was made based on the functional ECM framework [GHH+06] and includes: "ECM Access", "ECM Manage", and "ECM Govern" subclasses. "ECM Access" is a term that defines communication methods between enterprise and "ECM manage" classes. This class has associated Create Retrieve Update Delete (CRUD) operations and could support different manage classes. "ECM Manage" defines high level abstraction classes of management terms. The class has two subclasses that define core management and extension terms.
"Relationship" class describes relationships that can be established between classes. Each subclass of this class has "source" and "target" properties that define between which classes, relationship can be established.
"Policy" class describes corporate policies and quality of service requirements. The properties of the class were derived from quantitative characteristics of IBM Questionnaire.

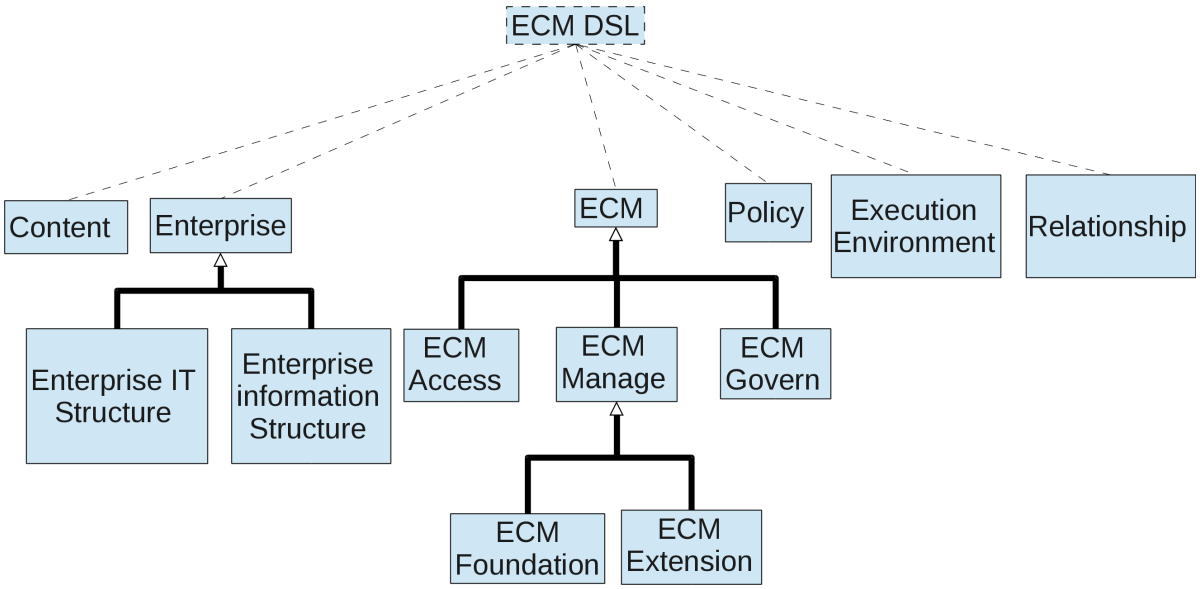

Figure 4.1: Simplified ontology class hierarchy

Enterprise class Enterprise class enables DSL user to define enterprise related properties that help to create ECMS architecture. ECMS requires information about enterprise for effective ECM. For example, records management is done according to the enterprise file plan, retention schedule, and information taxonomy. It has three direct subclasses: "Enterprise Information Structure", "Enterprise IT Structure" and "Disposal". Figure 4.2 shows the hierarchical structure of Enterprise terms.

"Enterprise Information Structure" describes the classification of enterprise information. "Information Taxonomy" term defines the classification scheme of information and its access control. It contains categories that help to structure content. For each type of the content the "Taxonomy Category" can be defined. "EnterpriseTaxonomy" class can have an unbounded number of references to "Taxonomy Category" class.
"File Plan" class defines the hierarchical structure and access control to records across the enterprise. The typical file plan consists of the hierarchy of categories and subcategories, and can have the following levels [HEI07]:

- Functions describe the responsibilities of organizations that help to fulfill its goals and usually does not represent organizational tasks [ZAB+09].
- Activities and sub-activities are major tasks of organization that allows to execute functions. One function may contain several activities. Each activity can be split into several sub-activities [ZAB+09].
- Records series unites the records that are related to a particular subject or function.

"File Plan" class allows business users to define the record structure that can be used for the records management purposes. It contains the "classification scheme" and "Record Categories" properties. "Record Categories" has the reference to "File plan Category" class as a

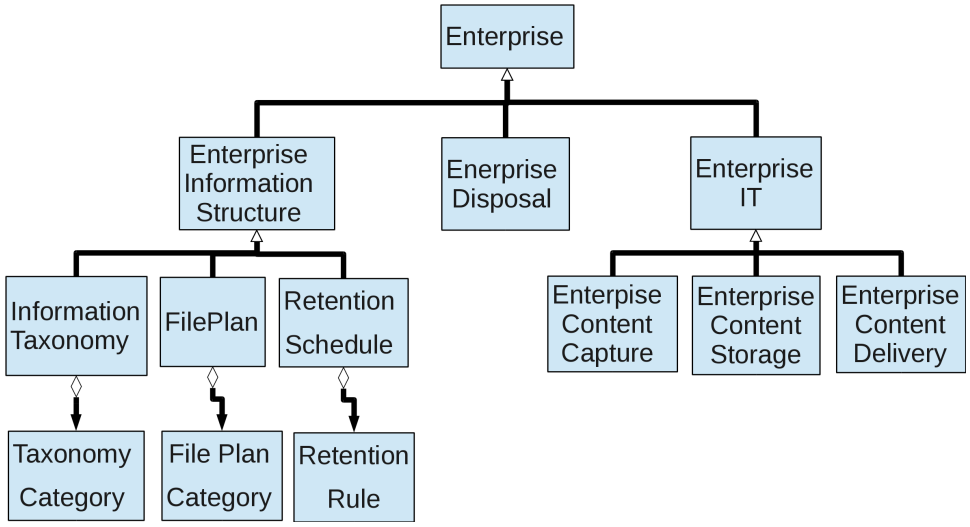

Figure 4.2: Hierarchy of enterprise classes

value that describes functions, activities, sub-activities and records series of file plan."
"Retention schedule" defines records retention rules. It is associated with the ECMS records management for definition of periods during which content should be preserved. When the retention period is expired the record should be disposed according to the disposal method defined by retention schedule rule. "Retention schedule" class has retention rule properties that have a reference to "Retention Rule" class as a value.
"Enterprise IT Structure" class is a generic class for all components that describe capture, store and delivery methods of content within the enterprise. This component allows to define capture requirements of ECMS. For example, a declaration of "Enterprise Content Storage" defines from which sources enterprise content should be collected. This information helps cloud provider to define correct content collection components depending on the places where the data is stored. For example, a definition of SMTP mail server requires a usage of collection component that has the possibility to collect the data from SMTP mail servers.
"Enterprise IT" class inherits properties from "Enterprise". "Enterprise Content Capture", "Enterprise Content Storage" and "Enterprise Content Delivery" classes are derived from "Enterprise IT" class. "Enterprise Content Capture" is a generic class that defines hardware and software acquisition and capturing methods. It has the following properties: "Manufacturer", "Version", "DocumentsCapturePerDay". "Manufacturer" defines the developer of software or hardware component. It describes, which capture manufacturers should be supported by ECMS. For example, ECMS imaging software can support drivers from different imaging system manufacturers. "Version" defines a revision of hardware component or release number of software component. "DocumentsCapturePerDay" defines the approximate number of documents that are acquired by the component. This property allows to define the total enterprise load when all capture components are defined. "Enterprise Content Storage" defines sources where the enterprise content is stored. It could be file or mail servers, different unstructured repositories, and others. This class has no own properties and can be extended by the storage subclasses (servers, repositories, cloud storage, etc.). An example of the extension is shown

on Figure 4.3.
"Enterprise Content Delivery" class defines content delivery methods within the enterprise.

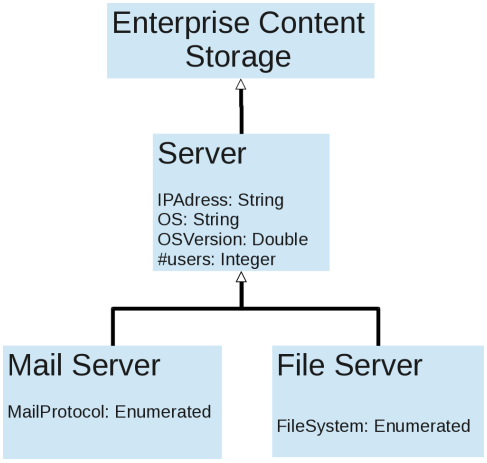

Figure 4.3: Example of Enterprise Content Storage Class Extension

Its instances can include software and hardware components: printers, viewers, etc. It is a generic class, from which concrete classes can be derived.
According to the ontology development guideline [NM], there is no need to define all possible classes of the domain, but only classes that will be used by application. Because of that, I defined only generic classes of the Enterprise IT Structure. There are some IT terms that could describe capturing and delivering of information. Such classes could be distinguished in a separate class that will be derived from "Enterprise Content Capture" and "Enterprise Content Delivery" classes.
"Disposal" term defines the final stage of ECM lifecycle. It describe actions, that should be applied to information after the expiration of retention period.

ECM classes The ECM related terms are represented by "ECM" class from which all other ECM classes are derived. Figure 4.4 shows the hierarchical structure of ECM classes.
"ECM" class "Interface" property describes operations that are associated with ECM terms. For example, the following basic ECM operations are associated with "ECM Repository": "Check-in", "Check-out", "Retrieve", "Update", "Delete", and others. The supported operations for each ECM term were derived from technical documentation of different ECM vendors. CMIS standard was also used to define operations.
The ECM terms are split logically on three categories: ECM Access, ECM Manage, and ECM Govern categories.
The access category is represented by "ECM Access" class and declares terms that describe capture and delivery of the content that is stored in ECMS. Different ECM frameworks present different definitions of the capture and delivery layers. [GHH+06] defines the access layer that unites all capture and delivery classes. [oT14] distinguishes capture and delivery classes as separate layers. I used the common term that describes properties of both types of layers because in ECMS one class (Web client, portal, etc.) can describe both capture

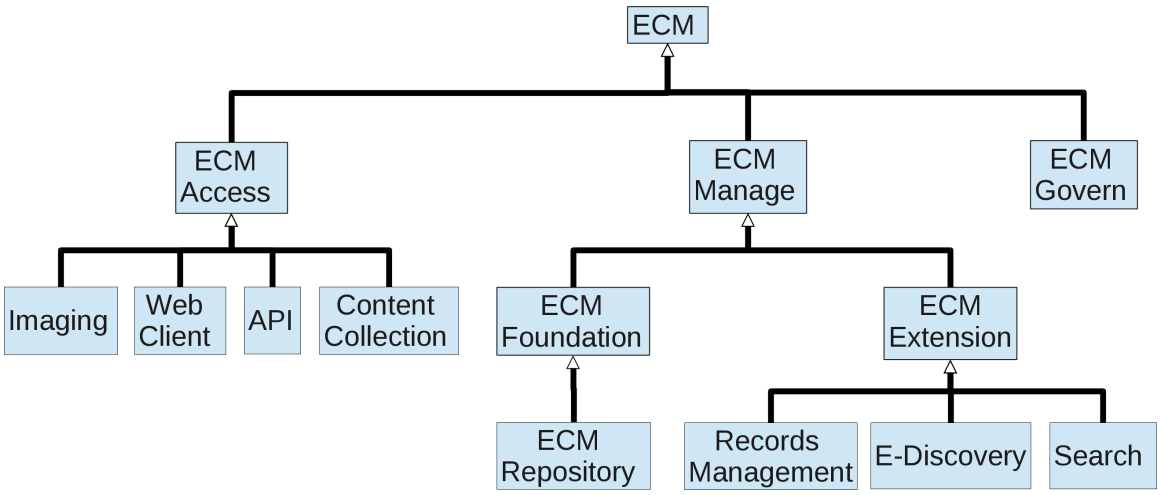

Figure 4.4: Hierarchical structure of ECM classes

and delivery functionality (exceptions are imaging and content collection terms that describe uploading of a content to the ECM repository).
The manage category is represented by "ECM Manage" class and describes management aspects of enterprise content. The Gartner's definition of ECM consists of two framework: a strategic framework and a technical architecture. The technical architecture is divided on two parts: a platform and ECM extensions. The platform defines core functionalities of ECMS. Extensions define additional components that extend basic functionality [GSC+14]. Analysis of ECM solutions from different vendors shows, that there are a basic components that form ECM foundation. All other components adds functionality to ECMS.
Based on Gartner's definition of ECM and ECMS architectures from different vendors, I decided to introduce two terms: "ECM Foundation" and "ECM Extension". "ECM Foundation" represents terms that are mandatory for the ECM language. All extension terms requires the definition of "ECM Foundation". For example, if a user of the language defines "Records Management", he also must define "ECM Repository", because "Records Managament" adds functionality to "ECM Repository".
"ECM repository" is derived from "ECM foundation" class. It describes a storage that preserves content and its metadata and library services that implements the basic ECM operations: create, retrieve, update, delete, search, check-in check-out, manage document life cycle and control access to information "ECM Extension" class defines terms that describe concepts of management of enterprise content but are not a part of the foundation. I defined "Records Management", "E-Discovery" and "Search" terms as children. Other terms can be defined as children of "ECM Extension" class. "ECM Extension" class has "extends" property that declares the dependency of this class from "ECM Foundation".
There are different factors that impact on content management: business, records management, legal, IT. Business have to capture, manage, analyze and properly dispose enterprise information. Records Management defines, which content should be declared as records and defines retention rules. Legal declares that the enterprise content should be available in a case of government investigation. "Records management" and "E-discovery" terms describe

concepts that help to mitigate the ECM risks.
"Records management" term defines the concept that describes initialization and maintenance of enterprise records. It uses a corporate file plan and retention schedule to define records structure and retention periods for the records series. The following record and retention management operations are associated with "Records Management" term: define a corporate taxonomy, create a file plan, define retention policies, declare documents as records, define disposition schedules, dispose of records, create disposition reports and keep an audit trail of all relevant operations to comply with corporate and legal retention rules.
"E-discovery" term defines requirements for a process of discovery in civil litigation that is carried out in electronic format.
"ECM Govern" term is a generic class. It declares the possibility to execute the following operation: retention policy and schedule management, e-discovery process management, disposal and governance management for IT. This component can be used as substitution in cases when the Enterprise structure is not provided.

Relationship class "Relationship" class defines classes that allow to link ECM terms. The following subclasses are defined (Figure 4.5): "capturedBy", "exportedTo", "flows", "collectedBy" "Supports", "appliedTo", "dependsOn", "extends", "requires".

Each subclass of the "Relationship" class contains the following properties: "source" and

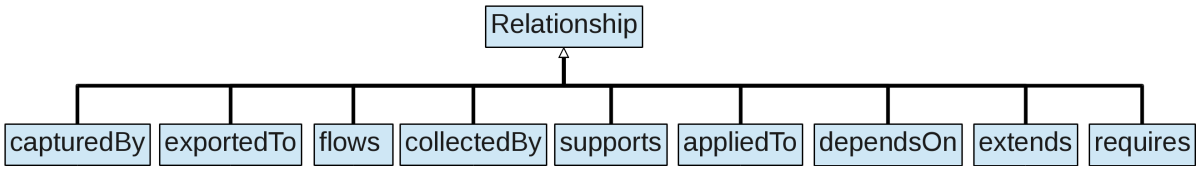

Figure 4.5: ECM ontology "relationship" class hierarchy

"target" that have references to the ontology classes as a value. These parameters define the direction of the relationship. "source" defines from which class the relationship is established. "target" declares a destination of the relationship.
"CapturedBy", "exportedTo", "flows", "collectedBy" defines content flow relationships within organization and ECMS. "CapturedBy" relationship can be defined between "Content" and "ECM Access" classes. It defines, how the content is acquired by ECMS. "ExportedTo" has "ECM Access" class as a source and "Content" class as a property. It defines to which formats content that is stored in the ECM repository can be exported. "Flows" class defines on which Enterprise IT resources the content is stored. It has "Content" class defined as a source and "Enterprise IT" class as a target property. "CollectedBy" class defines a link between Enterprise IT storage resources and "ECM Access" terms, and defines from which enterprise storage resources the data should be collected and stored in the ECM repository. Introduction of this relationship was derived from analysis of IBM technical documentation. IBM provides the possibility to collect content from customer's storage resources (e.g. mail or file servers). It requires the information about the enterprise data storages (e.g. protocols). "CollectedBy" relationship links enterprise storage resources that are defined by instantiation of "Enterprise IT" and "Content Collection" classes. "Content Collection" class is a subclass

of the "ECM Access" class.
"Supports" relationship can be established between "ECM Access" and "ECM Manage" classes and defines which operations that are associated with ECM manage terms should be also associated with ECM access terms. For example, if "Web client" (instance of "ECM Access" class) has "supports" relationship with an instance of "Records Management" class, then it means that operations that are defined for "Records Management" (e.g "declare record"), should be also supported by "Web client" (the user should has the possibility to declare records using web client). This relationship shows how enterprise content is captured and retrieved by ECMS.
"AppliedTo" relationship defines to which terms should be applied a policy. This relationship can be declared between "Policy" class and all classes that are inherited from "ECM" class. It defines "Policy" class as a valid value for the "source" property, and "ECM" class for the "target" property.
"DependsOn" relationship defines dependencies between ECM and execution environment. It declares that an instance of "ECM" class requires a presence of the instance of "Execution environment" class. The following relationship can be established: "ECM repository" depends on "Database" and "Storage". For cloud provider this information means that the storage and the database with defined properties should be deployed before the ECM Repository deployment. "DependsOn" class defines "ECM" class as a valid value for "source" property, and "Execution Environment" class as a "target" property.
"Extends" relationship can be defined between "ECM extension" and "ECM Repository" instances. It declares that instance of "ECM extension" class adds functionality to the "ECM repository". This relationship was derived from an examination of the ECM solutions from different vendors. The analysis showed, that ECMS consist of platform component, that is defined by a storage, databases, and library services that allow to store enterprise content and its metadata and execute operations (create, retrieve, update, delete, etc.) on it, and the extension components that define additional operations that are associated with an extension component.
"Requires" relationship defines dependencies between "ECM extension" and "Enterprise" classes. It declares that an instance of "ECM Extension" requires information about the enterprise. This class defines "ECM extension" as a "source" property, and "Enterprise" class as "target" property. This class can be extended to provide more concrete relationships between ECM extensions and an enterprise.
Complete ECM Ontology including relationships between classes is shown on Figure 4.6. Blue boxes declare ontology classes that define ECM DSL terms. Relationships between terms are represented by ontology classes that have source and target references (green nodes and edges).

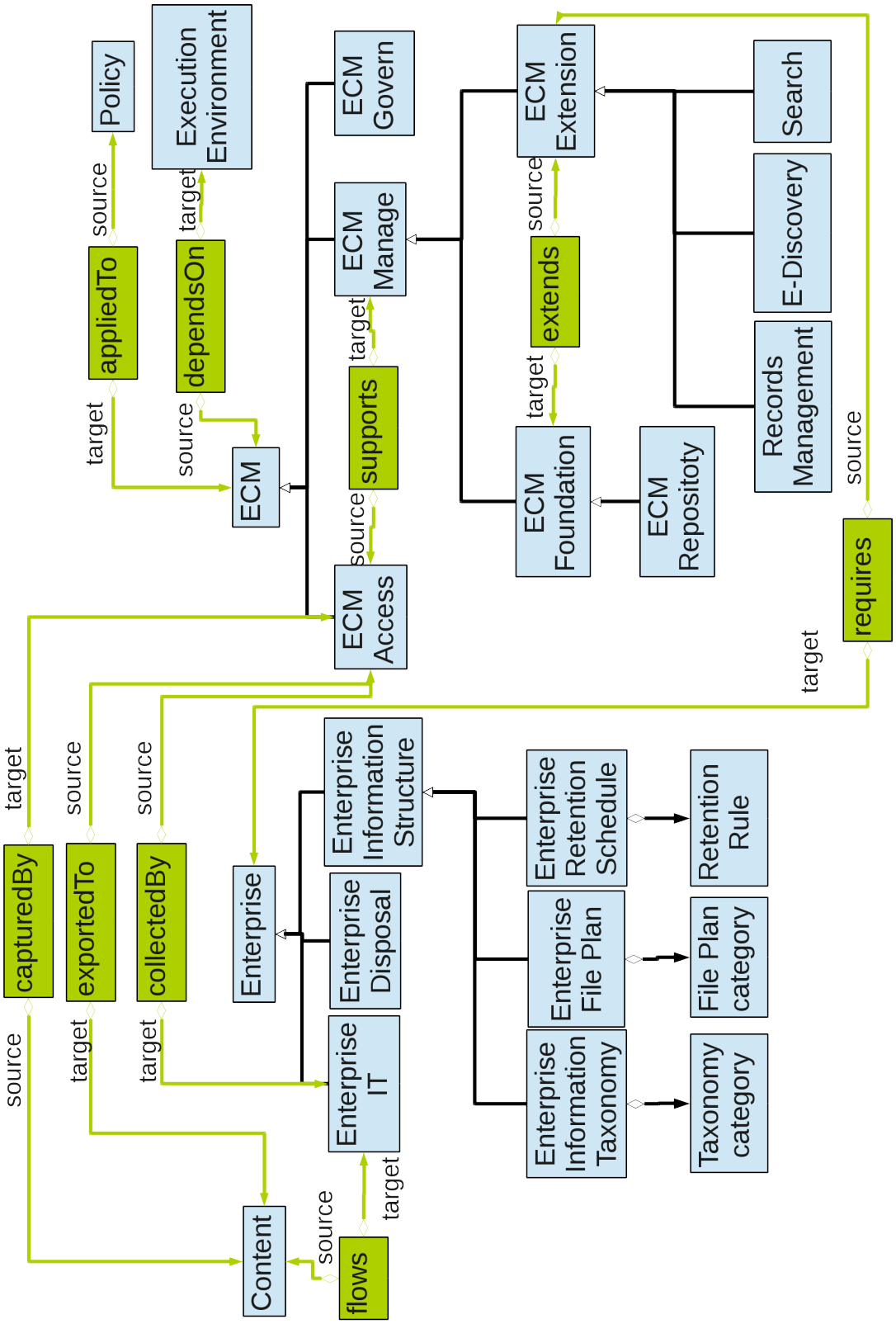

Figure 4.6: ECM ontology class hierarchy with relationships

4.2 Design of ECM DSL

This section describes abstract and concrete syntax based on the developed ontology. The abstract description of the language was done in informal way by providing an example. Concrete description was done using TOSCA language.

4.2.1 ECM language description

The goal of the DSL is to create high-level topology of ECM solution and to declare functional and non-functional requirements of it. The functional requirements define basic operations that ECM user wants to perform, and include qualitative characteristics of the system. The qualitative parameters define type of components and their workloads (list of operations that are supported by component). The non-functional requirements define quantitative characteristics and policies that should be applied to ECMS. Quantitative parameters include information about customer's workload to the system and define the number of concurrent users, average document size, maximum and minimum create, update, retrieve, delete and search operations. Policies indicate availability, business continuity, target application, performance characteristics of ECMS and should satisfy business and legal requirements. Business requirements could be defined by SLA and could contain main KPI indicators. Legal requirements define parameters that describe application of ECMS according local litigations (e.g. location of the system, or disaster recovery parameters).
Resulted DSL defines syntax and semantics and allows users to define requirements to ECMS. Semantics define the vocabulary of the DSL and according to the developed ontology is defined by classes and subclasses. Syntax defines rules that describe how the components are related to each other.
The main users of the DSL are business users, ECM architects, and Cloud providers.
Business users define functional and non-functional requirements to ECMS. ECM architects create ECM topologies using ECM language and define the parameters that should be provided by business customers.
Cloud provider translates ECM topology to the particular architecture. The abstract syntax of the ECM language represents an informal description by providing an example.
As a basis of the example the following use case was used: "As the Business User I want to create a document in the repository and apply version control. Additionally I want to store all the documents from enterprise storage in the repository. As a Record Manager of my enterprise I want to define the corporate taxonomy being able to classify documents and business relevant information into mandate corporate record categories."

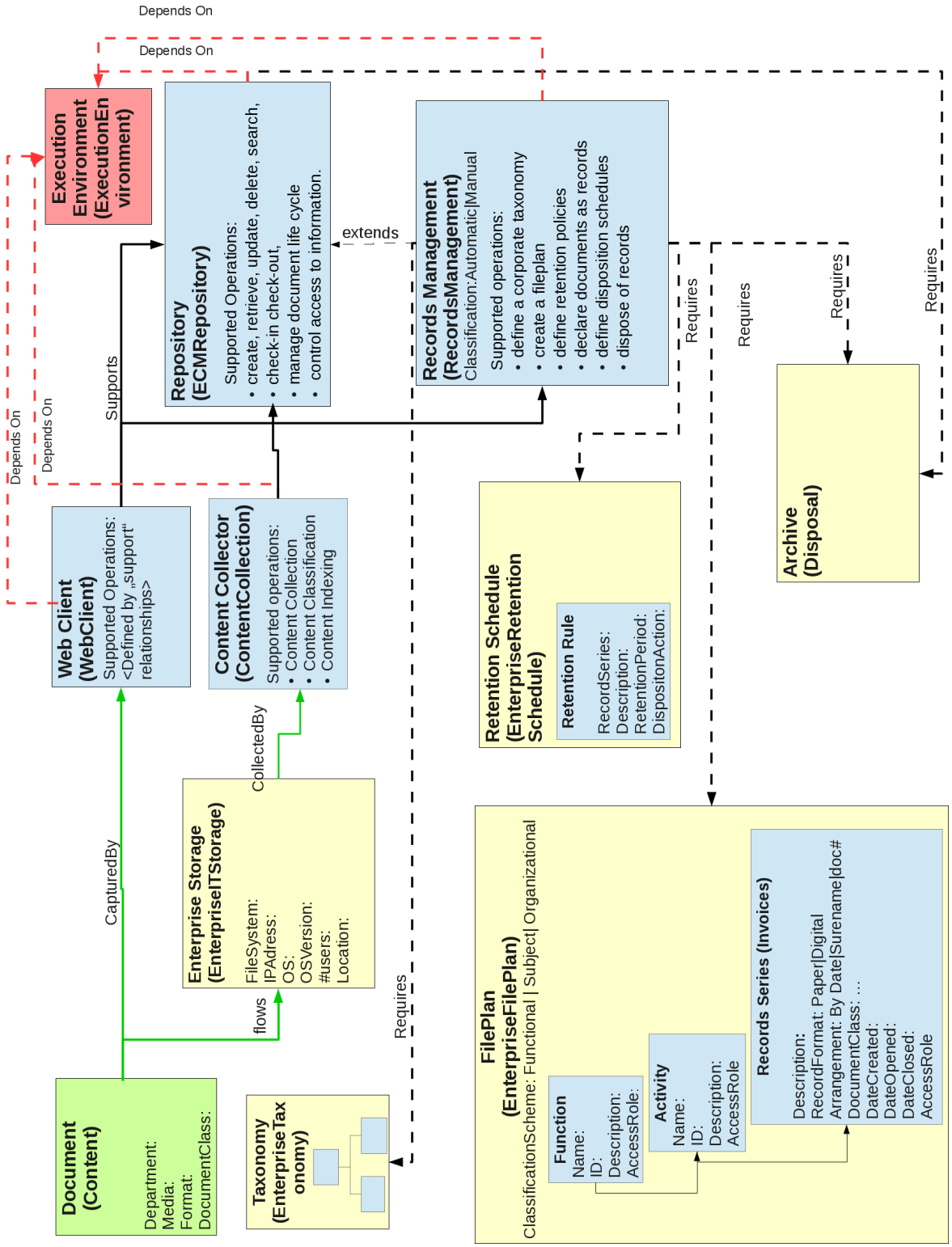

Figure 4.7: An example of the DSL topology

Based on the provided use case, ECM architect models the ECM topology. The example of te topology of DSL terms and relationships between them is shown on figure 4.7. The topology represents a graph, which vertices are represented by instances of the content, ECM and enterprise classes. The edges of the graph are represented by the relationships between nodes. Green nodes describe content, that defines documents created or acquired by enterprise. Yellow nodes describe enterprise related terms. Blue nodes define ECM related terms and red node describe ECM execution environment.
At the basis of ECM DSL lies the content lifecycle. Content describes the different types of enterprise content and defines its characteristics: format, type, name of department that creates or captures content. Green line on Figure 4.7 shows the content flow within enterprise. An example shows that the "Document" (instance of the content class) can be stored on "Enterprise storage" node or can be directly uploaded to the ECM system using ECM web client. The content stored on "ECM Storage Node" should be collected using ECM content collection instance. Figure 4.7 shows only one case of the content flow within enterprise. The description of the content creation can be shown more detailed. Subclasses of "EnterpriseIT" class allows to create instances that allow to describe capture, retrieval and store enterprise components. For example, user can define that the document was created by "Word Processor" and then stored on "Enterprise Content Storage" node. "Enterprise Content Capture", "Enterprise Content Storage" and "Enterprise Content Delivery" classes could describe software and hardware components that are used to capture, store and deliver of content within enterprise.
"ECM Access" class and its subclasses are used as boundary components between enterprise and ECM Manage classes. They are used to enable ECM related operations and can support operations of ECM manage components. "Support" relationship is used to link access and management terms. For example, "Web client" supports "Repository" and "Records Manager" components. It means, that "Web client" allows to perform operations that are supported by these component (e.g. create, retrieve, delete, search content, declare records, etc.). Each ECM node provides two types of interfaces: user interface, that unites the operations that can be executed by the end user, and administration interface, that define provisioning and deprovisioning operations.
End user interface allows cloud provider to choose correct ECM components based on the workloads that are associated with ECM terms.
Examination of the technical documentation from different ECM developers showed that all ECM solutions have the same principle: there is a core component that implements basic content management operations and provides repository for content and metadata. All other components extend functionality of the core component or connect to this component using API. According to this information, the ECM DSL has the following structure: Repository (instance of "ECMRepository" class) declares the term that defines mandatory ECM functionality. All other terms define additional functionality and require the presence of the repository. The restriction that describes dependencies between core and extension ECM nodes should be defined. This restriction should not allow to introduce ECM extension term without introducing core ECM term ("ECM Foundation"). The validation of the topology could be done before its execution. For example, the topology is incomplete if "Repository" is not presented.
All ECM components depend on execution environment. This term describe components that

should be deployed before the ECM component deployment. This term can be substituted by the low level components during a translation step of the high level topology to the low level architecture, that defines components from ECM vendors. For example, "Repository" node depends on the database, file system and text index components. In that case the execution sequence will be defined as follow:

1. Rational database should be installed, database schema and tables should be created.
2. Storage with appropriate file system should be allocated.
3. Text index engine should be installed;
4. Repository component should be installed
5. Extension and access components could be installed;

ECM components may also require the information about enterprise architecture. For example, "records management" requires information about the corporate file plan for records categorization and retention schedule for the records disposition.
There are two possibilities to provide enterprise architecture. The user can provide the information about corporate taxonomy, file plan, and retention schedule, or he/she can define the "ECM govern component" that indicates that ECM solution should support the interface for enterprise parameters description.

Relationships between instances Relationships between instances of the ECM ontology classes define dependencies between them. There are three types of relationships: Content flow, compatibility and dependency relationship types.
"Flows", "capturedBy","exportedTo", "collectedBy" form content flow relationships and define dependencies between instances of the content, enterprise, and ECM access classes. It allows to define the content properties that are captured by "ECM access". For example, the user can link several content nodes with web client. It means that content of the declared type is directly uploaded to the ECMS. The following types of information could be derived by analyzing content flow relationships: supported content formats, the average size of the content object, etc. Additional properties could be defined by ECM architect or cloud provider depending on information he/she needs for ECMS deployment.
Establishing "flows" relationship between "Content" and "Enterprise Content Capture" node defines which types of content are captured (delivered) using access component.
Defining "flows" relationship between "Content" node and "Enterprise Content Storage" node declares, which type of content is stored on storage component.
Defining "capturedBy" relationship between content node and "ECM Access Component" declares which types of content can be uploaded directly to the ECMS using ECM Access nodes. Creating content flow relationships between enterprise and ECM access nodes defines that export/import of the defined types of content should be supported by enterprise software using appropriate ECM access node. For example, enterprise can have already existed software that supports CMIS interface, and it wants to use this software to upload/retrieve enterprise content to ECMS. Then a service topology should define content nodes, enterprise

access node that will be connected to ECM API node.
Declaring "collectedBy" relationship between "Enterprise Content Storage" node and ECM "Content Collection" node defines, from which resources enterprise content should be collected. Compatibility relationships are defined on Figure 4.7 by "supports" interactions between ECM access and manage nodes. They define operations of which nodes should be supported by ECM access node. All operations that are associated with an instance of the "ECM manage" class should be also associated with an instance of the "ECM access" class that has "supports" connection to that manage node. For example, the "Web Client" component has to support "Repository" and "Records Management" component. It means that the user should have the possibility to perform repository and records management operations defined by these components through ECM web client. There is a possibility to add compatibility relationships between any ECM access and management nodes.
Dependency relationships define dependencies between ECM components and between ECM and Enterprise components. This type of the relationships is presented on Figure 4.7 by "Requires", "DependsOn" and "extends" relationships.
"Requires" relationships are established between ECM and Enterprise nodes. This dependencies indicate that the ECM node needs information about enterprise. For example, "Records Management" node requires information about corporate file plan, retention schedule and disposition methods.
It allows to validate completeness of the topology. For example, if the ECM architect defined only "Records Management" component and does not specify the file plan, then the topology will not be executed, or execution engine has to complete topology automatically.
"Extends" relationships are established between foundation and extension ECM nodes. This dependency indicates that the extension node adds functionality to the instance of "ECM foundation" class and requires it.
"Depends on" relationship defines dependencies between instances of ECM and execution environment classes. The dependency indicates that ECM node requires the presence of execution environment node. An execution environment node could describe infrastructure components that are required for ECMS deployment. The use case defines only definition of qualitative requirements. Additionally, the instance of the policy class can be defined and "appliedTo" relationships between policies and instances of "ECM" subclasses can be established.

Topology design use case The typical use case of ECM topology design and execution is shown on Figure 4.8 and is defined as follows:

1. Business user defines functional and non-functional requirements of ECMS and sends them to ECM architect. The requirements can be defined as "request for proposal" document.

2. ECM architect defines ECM language terms and actions that are associated with these terms, creates a service template and publishes resulted topologies in the specialized catalog.

3. Business user defines types of enterprise content and content flow within enterprise. He/she also fills the properties of the topology nodes and policies. After that user sends

the resulted topology to cloud provider.

4. Cloud Provider deploys the ECM solution according to the received topology. Cloud provider translates the high level topology to the low level components. He could execute topology by using created adapter that validates the topology, derives qualitative and quantitative parameters from service topology and deploys ECMS system according derived information. The choice of the exact nodes can be made based on the operations that should be supported by the ECM high level nodes. If operations, requirements and capabilities that are supported by the high level terms coincide with operations, requirements and capabilities supported by the exact component, then the high level node can be substituted by the exact component.

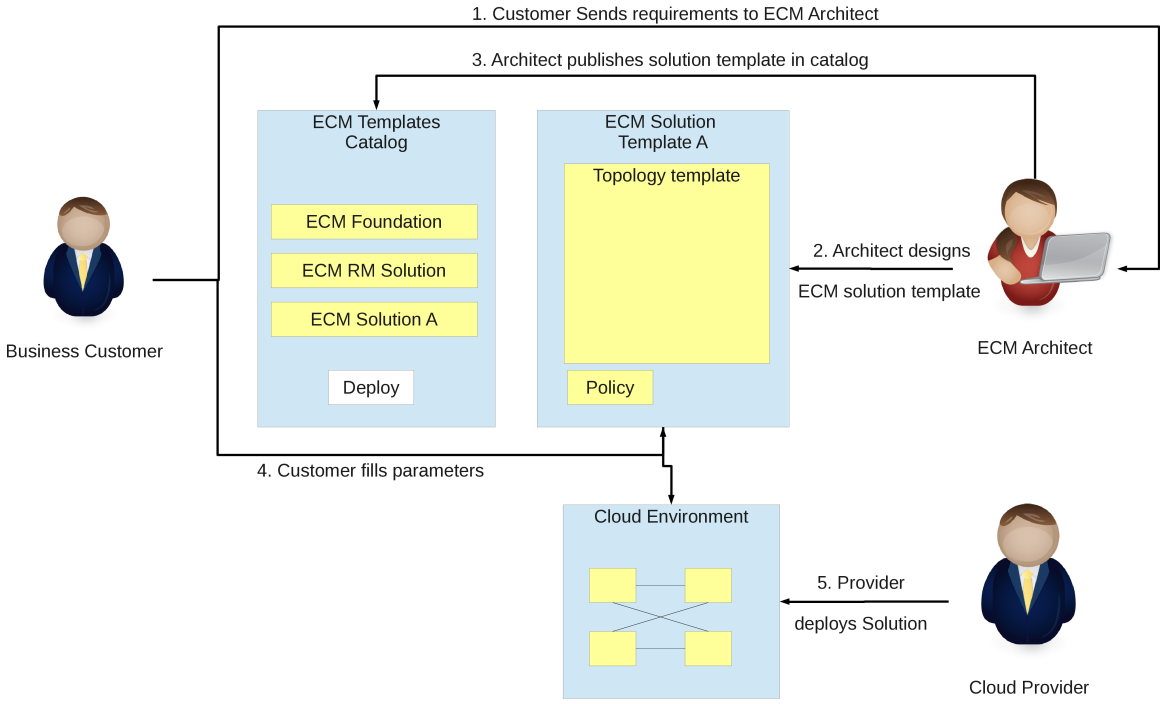

Figure 4.8: Topology deployment scheme

4.2.2 TOSCA implementation

An implementation of ECM DSL using concrete syntax was done by the designing TOSCA node types, requirements and capabilities, relationships, and policies. The goal of the language implementation using TOSCA is to allow user of the language (ECM architect, cloud provider or business user) to develop a service template that will define the functional and non-functional requirements of ECMS. The functional requirements should define workloads that ECMS should support. The non-functional requirements should specify criteria that allows to evaluate performance of ECMS (e.g. maximum response time, availability, etc.). Cloud provider have to satisfy defined performance criteria. For these purposes, business user

have to provide quantitative characteristics of the ECMS load (e.g. maximum concurrent users, average content size, locations).
The developed ECM ontology is used as a basis for the DSL concrete syntax. The ECM ontology defines the classes and properties that describe ECM and enterprise terms and relationships between them. The implementation of the ECM language was done by mapping ontology classes and properties to TOSCA node types. The ontology relationship classes were mapped to TOSCA relationships. "Source" and "Target" properties were substituted by the "validSource" and "validTarget" properties of the TOSCA relationships.
The ECM language description section defines the statement that the designed ECM topology should be complete (all required components should be presented). Requirements and capabilities were used to describe dependencies between ECM nodes. According to the TOSCA specification [Tos13b] each node requirement should be satisfied by appropriate capability. It allows to verify service template completeness.

Node types definition Node types hierarchical structure is shown on Figure 4.9. The definition of the ECM node types was done by mapping the following classes and their subclasses to the TOSCA node types: "Content", "Enterprise", "ECM", and "ExecutionEnvironment". For the "Content Node", "Enterprise Node", and "ECM Node" types the parent "Main Node" node type was defined. This node type describes the main concepts of the language. For "Execution Environment Node" the "Supplementary Node" type was introduced.
The ontology methodology supports multi inheritance for classes. The TOSCA specification allows to have only one parent for each node type [Tos13b]. According to this statement, enterprise content capture and delivery classes were united in one node type: "Enterprise Content Access Node". It combines properties of the capture and delivery nodes. If an instance of this node type implements capture and delivery content functionality, then all node type properties can be used. If the instance implements only capture (delivery) of content, then only capture (delivery) properties should be filled, and delivery (capture) parameters can be omitted. The same scheme is defined for ECM access nodes. To support such functionality the minimum occurs for property elements were set to 0.
"Taxonomy Category", "File Plan Category" and "Retention Rule" classes defines containment classes of "Enterprise" subclasses in the ECM ontology. In TOSCA these classes were replaced by the properties of "Enterprise" nodes. For example, ECM ontology "Enterprise Retention Schedule" class has the containment reference to "Retention Rule" class, that define retention rules. In TOSCA an implementation of this reference was replaced by a "Retention Rule" property element of "Enterprise Retention Schedule" node.
Node types have hierarchical structure. Each node type(except Root Node) inherits properties from parent node type. The inheritance behavior is defined by the TOSCA standard [Tos13b]. The following node parameters are inherited: node type properties, requirements and capabilities, interfaces.

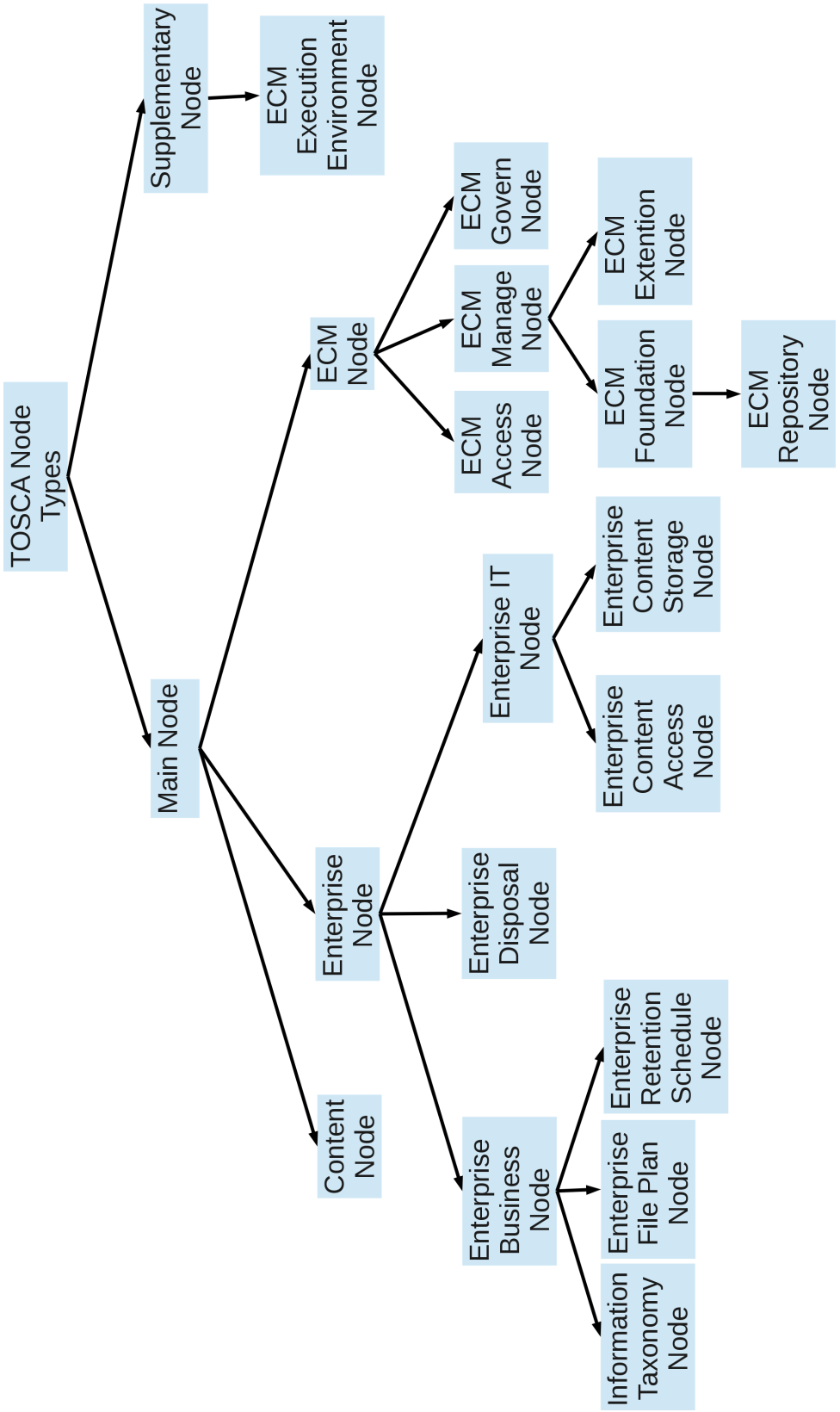

Figure 4.9: ECM node types hierarchy

The structure of the node type is defined by TOSCA standard and is shown in the following listing:

```
<NodeType name="xs:string">
        <documentation> xs:string </documentation>
        <DerivedFrom typeRef="xs:QName"/>
        <PropertiesDefninition element="xs:QName"/>
        <RequirementDefinitions>
                <RequirementDefinition
                        requirementType="xs:QName"
                        name="xs:string"
                        lowerBound="xs:integer"
                        upperBound="xs:integer"/>
        </RequirementDefinitions>
        <CapabilityDefinitions>
                <CapabilityDefinition name="xs:string"
                         capabilityType="xs:QName"
                         lowerBound="xs:integer"
                         upperBound="xs:integer"/>
        </CapabilityDefinitions>
        <Interfaces>
                <Interface name="UserInterface">
                        <Operation name="xs:string" />
                </Interface>
                <Interface name="AdministrationInterface">
                        <Operation name="xs:string" />
                </Interface>
        </Interfaces>
</NodeType>
```

DSL defines two interfaces for ECM nodes: user and administration interfaces. User interface defines operations that are supported by ECM element. These operations describe ECM component workload. This interface is defined by DSL to describe which functionality is supported by components. The interface can be extended by the introduction of a new node type and deriving properties from the parent node type. The new node type derives interfaces from the parent node type. If the new node type declares same interfaces, then these interfaces overwrite parent interfaces [Tos13b]. In that case the list of the operations that represents operation of parent node and introduced node should be defines. For example, if the "ECM Repository" contains "User Interface" with following operations: "create", "delete", "retrieve", "update", and the new node type "Custom ECM Repository" is introduced that adds additionally "versioning" operation to user interface, then the new node should define also parent operations for "User Interface". "User Interface" for the new node type will be defined as follows: "create", "delete", "retrieve", "update", "versioning". Cloud provider uses user interface during mapping high level nodes to particular components. The particular ECM component can be deployed if its user interface coincides with DSL node interface.For example,

"ECM Repository Node" defines the following user interface operations: createDocument, retrieveDocument, updateDocument, deleteDocument, searchDocument, checkIn, checkOut. Vendor's repository component (or set of components) should support these operations to be able to substitute the "ECM Repository Node".
The administration interface defines operations for the component de/provisioning.
"ECM Execution Environment Node" is a generic node type, from which more concrete nodes can be instantiated. For example, before the repository can be deployed, the appropriate environment that contains installed rational database, allocated storage and text index, should be deployed. An example of the new node types that are derived from "ECM Execution Environment Node" is shown on Figure 4.10. Node Types marked with green color represent

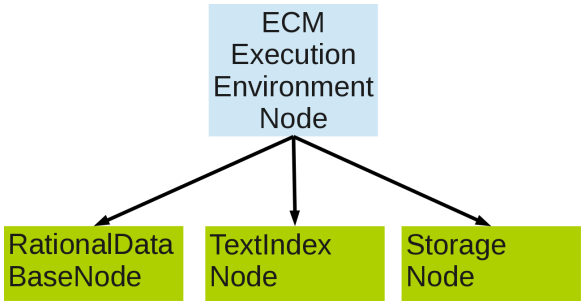

Figure 4.10: ECM execution environment node example

components created by ECM Architect to define execution nodes for ECM Repository.

Requirements and capabilities Requirements and capabilities define dependencies between nodes. Their hierarchical structures are shown on Figures 4.11 and 4.12.
Requirements and capabilities types were designed based on the "source" and "target" properties of the ECM ontology relationship classes. These properties define the references to the ontology classes and declare between which classes the relationship can be established. According to the value types of these properties requirements and capabilities were defined. For example, the ontology "dependsOn" relationship can be established between "ECM" and "ExecutionEnvironment" ontology classes. "ExecutionEnvironmentRequirement" and "ExecutionEnvironmentCapabilities" types were created to support "dependsOn" relationship in TOSCA notation.
Requirement and capability types have hierarchical structure. Each type(except Root Types) inherits properties from the parent type. The inheritance behavior is defined by the TOSCA standard [Tos13b]. The following requirement and capability parameters are inherited: requirement and capability type properties.
By default lower and upper bounds of the requirements and capabilities are set to 1. The lower bound of Requirement that equals one indicates that requirement must be fulfilled by appropriate capability and must be instantiated in Node Template. If the lower bound of the requirement equals 0, then the requirement is optional. The lower bound for capability must be more than 0.
The requirements and capabilities in ECM DSL are used to show dependencies between nodes,

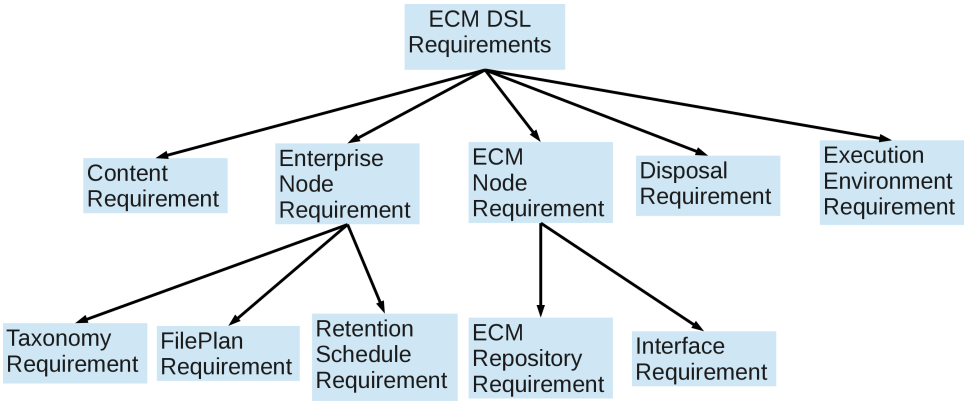

Figure 4.11: ECM DSL requirements types

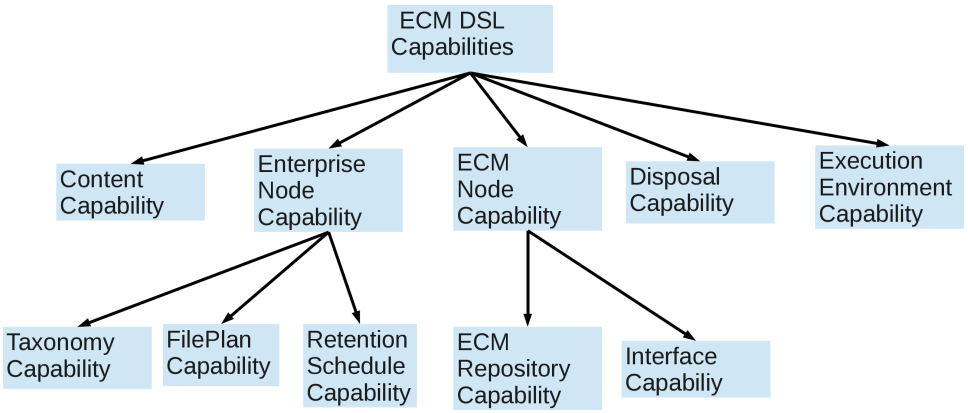

Figure 4.12: ECM DSL capabilities types

to define which relationships between nodes can be established, and for topology verification. According to the TOSCA standard each requirement should be fulfilled by the appropriate capability [Tos13b].
Content requirement and capability define between which nodes content flow can be established. Content Container Requirement is applied to the following node types: "Content Node", "Enterprise Access Node", "Enterprise Content Storage Node". It declares that the node type should be connected with the node type that can accept content. Content capability is applied for the following nodes: "ContentNode" "EnterpriseAccessNode", "EnterpriseStorageNode", "ECMAccessNode". This capability describes that the node type can accept content objects.
Enterprise node requirement and capability define the dependency between the ECM and enterprise nodes. For example, "Records Management" requires information about corporate file plan and retention schedule, and "Records Management node" type contains appropriate requirements that can be fulfilled by File plan ("EnterpriseFilePlanNode" type) and retention schedule ("EnterpriseRetentionScheduleNode" type) capabilities. "ECM Repository requirement" and appropriate capability define the dependency of ECM extension components

from Repository component. "ECM Repository Requirement" is applied to "ECM Extension Node" type and is inherited by all children of this node type. "ECM Repository Capability" is applied to "ECM Foundation Node" type and declares, that the repository functionality can be extended by another nodes. "Disposal requirement" and appropriate capability define dependency between ECM node types and Disposition methods that should be applied to a content according to the enterprise retention schedule. "Interface Requirement" and appropriate capability define between which nodes "Supports" relationship can be established. "InterfaceRequirement"is applied to "ECM Access node" type. This requirement is not mandatory and its lower bound equals 0. Upper bound equals "unbounded". It means that the ECM access nodes can support interfaces of different ECM manage nodes. "Interface capability" is defined for ECM manage nodes. Its lower bound equals 1 and upper bound equals "unbounded". It means that node supports interface capability.
"Execution Environment Requirement" and appropriate capability define dependencies between ECM node and terms that represent execution environment. It means that execution environment node has to be deployed before ECM component deployment

Relationships definition Relationships between nodes represent instances of the TOSCA relationship types. The ontology classes that represent link between terms were mapped to TOSCA relationship types. "source" and "target" properties were substituted by "validSources" and "validTargets" parameters that are defined by the TOSCA standard [Tos13b]. Ontology "Relationship" classes use references to other classes as values. TOSCA relationship types define source and target nodes by providing valid requirements and capabilities. "capturedBy", "exportedTo", "flows", "CollectedBy" ontology relationship classes were mapped to "ContentFlow" relationship type. "Supports", "dependsOn", "extends" and "requires" relationship ontology classes were mapped to the TOSCA types using the same names. "AppliedTo" relationship class was not mapped, because it defines dependencies between "Policy" and "ECM" ontology classes, and TOSCA provides own definition for Policy type.
In the ECM language relationships define not only dependencies between nodes, but are also used for the analysis of the ECM requirements. For example, "Supports" relationship declares, which ECM manage node interfaces should be supported by "ECMAccessNode". The hierarchical structure is shown on Figure 4.13
The relationship types have hierarchical structure. Each type(except Root Types) inherits

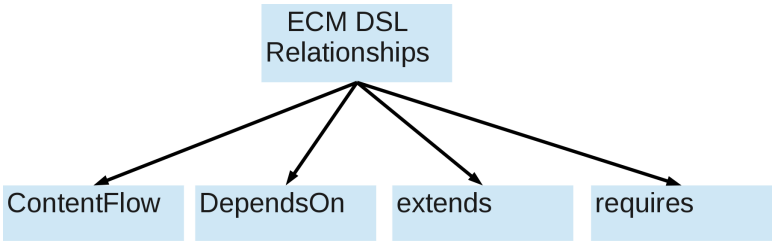

Figure 4.13: ECM DSL relationship types

properties from the parent type. The inheritance behavior is defined by the TOSCA standard [Tos13b]. The following relationship parameters are inherited: relationship type properties,

valid source and target, Interfaces. "Valid source" parameter defines the requirement type, and "valid target" parameter defines capability type. These two parameters are optional and define relationship restrictions: the relationship can be created only between nodes that contain valid requirement and capability types.
"ContentFlow" relationship defines the flow of content within enterprise. TOSCA provide reuse of the relationship types [Tos13b]. "flows", "capturedBy", "exportedTo", and "CollectedBy" ontology relationships were substituted by "ContentFlow" relationship. It allows to define which type of content, on which enterprise resources is stored, and it defines the way, how the content have to be uploaded to ECMS. "ContentFlow" relationship type defines the connection between Node types that contain "Content Container Requirement" and "Content Container Capability".
The example of relationship establishment is shown on Figure 4.14. The Content Flow rela-

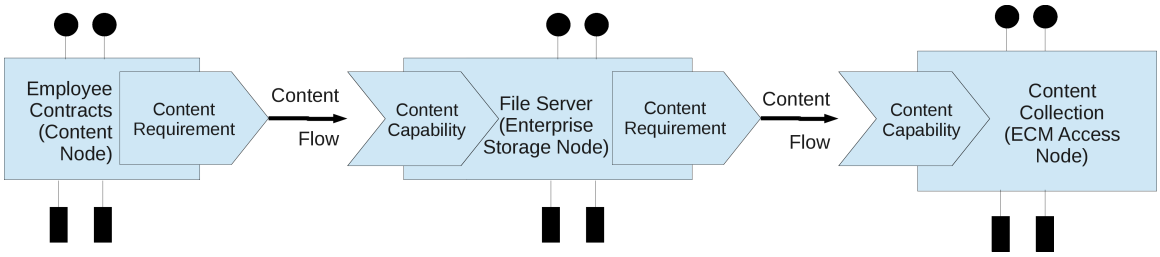

Figure 4.14: ECM DSL content flow relationship example

tionship between "Employee Contract" and "FileServer" indicates that "Employee Contract" content type is stored on "FileServer". Content Flow relationship between "FileServer" and "ContentCollector" defines that the content is collected from "FileServer" using "Content Collection" component. The analysis of that graph could provide the following information:

- Types of the content that should be collected and stored in ECMS;
- The way how the content is uploaded to ECMS (e.g. using Content Collection, Web Client, ECM API);
- From which enterprise resources the content could be collected
- Quantitative parameters of the content (e.g. average document size)

"DependsOn" relationship defines connections between ECM nodes that contain "Execution Environment Node Requirement", and Execution Environment nodes that contain "Execution Environment Node Capability". It allows to define which environment components (databases, storage, etc.) are used by the ECM node. Figure 4.15 shows an example of "DependsOn" Relationships. This examples defines that "ECM Repository" requires a definition of "Rational Database", "TextIndex" and "Storage". Each of ECM and "Execution Environment" nodes provides administration interface that can be used to define, which provisioning operations are supported by each node. "Extends" relationships are established between ECM extension and repository nodes. This relationship defines that extension node extends repository functionality and depends on it. This relationship can be established between nodes that contain

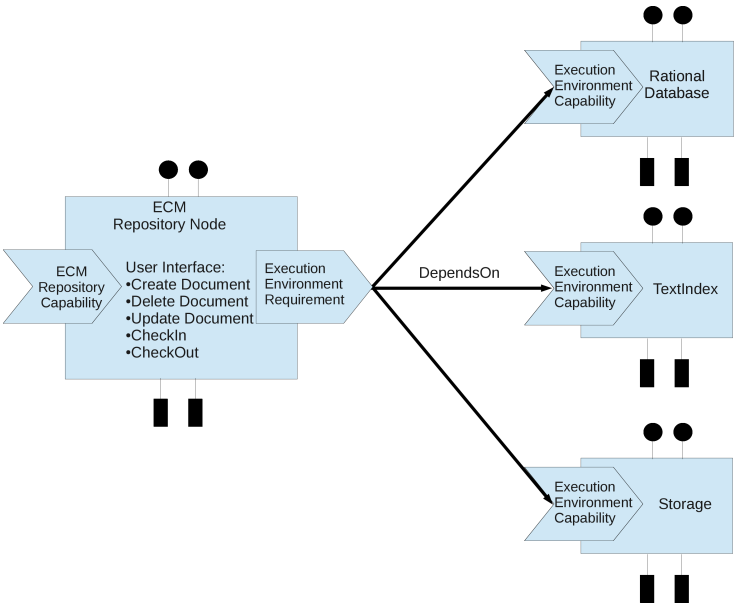

Figure 4.15: ECM DSL depends on relationship example

"ECM Repository Requirement" and "ECM Repository Capability". One repository can be extended by the different extension components.
"Supports" relationship can be created between ECM Access and manage nodes. It indicates that ECM access node supports interface operations provided by the ECM manage node. This relationship has "Interface Requirement" and "Interface Capability" as valid source and target parameters. For example, if the "Supports" relationship is established between Web Client and Repository, then it means that it should be possible by the user to execute repository operations using the web client (Figure 4.16).

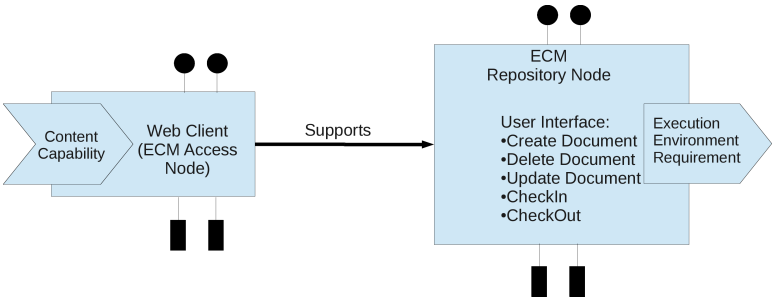

Figure 4.16: ECM DSL supports relationship example

"Requires" relationships define that the ECM node requires the information about the enterprise business structure (Taxonomy, File Plan, etc.). This relationship can be established between node types that contains "EnterpriseNodeRequirement" and "EnterpriseNodeCapability". ECM Govern component supports "EnterpriseNodeCapabity". Presence of this components indicates that business structure will be provided by the user after ECMS de-

ployment.
In this section I defined requirements and capabilities for all TOSCA relationship types. To evaluate the behavior of the DSL user, I omit the definition of requirements and capabilities for the following relationship types: "Content Flow" and "Depends On". It allows to evaluate, how DSL user uses connection during early iteration stages of the ECM language. "Extends" and "Requires" relationships are mandatory, because there are node types that requires a presence of these relationships (e.g the file plan and retention schedule should be associated with "Records management").

Policies definition Policies in the ECM ontology were defined by the introduction of "Policy" class and creating a "appliedTo" link between it and ECM classes. The TOSCA standard supports policy type that allows to define policies and assign them to node type [Tos13b]. The exact values should be provided in "Policy Template", and the reference to it should be defined for each ECM node template.
ECM language declares "ECM Policy Type" that is applied to ECM Node. It allows to define corporate policies, quantitative properties that will be need during ECMS deployment. The following properties can be defined:

- "Availability" defines the time during which the ECMS is not reachable by user.
- "Target Environment" defines the purposes of ECMS deployment (e.g. "DevelopmentEnvironment", "ProductionEnvironment").

Additionally, quantitative parameters can be declared. The final ECMS architecture depends on policy properties. For example, availability parameter impacts on type of components that should be deployed. "Target Environment" impacts on number of components that should be deployed. Additional parameters (e.g. "location") can be added by the ECM architect. Policy parameters can be defined by the business customer or could reflect requirements of the law.

Service template definition The output of ECM DSL is a service template that represents the high level description of the ECM and enterprise terms and defines the functional and non-functional requirements of ECMS. Policy, node and relationship templates should be declared in the service template. The listing provided in Appendix B shows an example of ECM foundation service template.
The foundation defines the mandatory nodes. For example, IBM ECM solution defines Content Navigator (Web Client) and P8 Server (ECM repository) as mandatory components. All following declarations are put in "Definitions" document:

- The policy templates instantiate policy types and contain values provided by the business users
- The boundary definitions provide user Interface operations that should be supported by ECMS system. The given example define repository operations.

- The topology template defines nodes instances, their properties and relationships between them.

Figure 4.17 shows the schematic structure of the ECM foundation service template. The

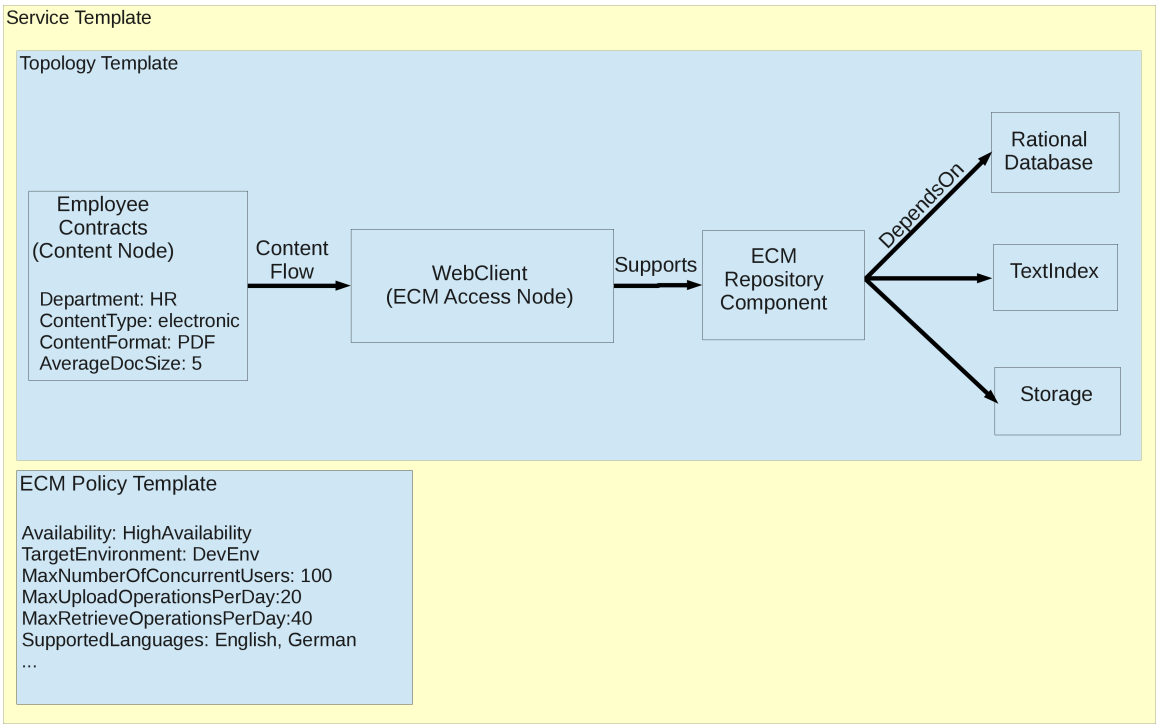

Figure 4.17: ECM DSL foundation scheme example

analysis of the requirements can be done by cloud provider by parsing the resulted service template.

DSL output The resulted service template as well as type definitions have to be included in CSAR archive. CSAR archive has the following structure (Figure 4.18):

- "Definitions" folder contains ECM DSL definitions (Node, Relationship, Policy, Requirement, Capability types).
- "Types" folder contains properties for node and policy types.
- "ECMPoliciyProperties.xml" contains xsd schema for the policy properties.
- "ECMBaseTypes.xml" contains Node, Relationship, Requirements, Capabilities, Policies types definition of ECM DSL that define the basis of the language.
- "ECMSpecificTypes.xml" contains Node, Relationship, Requirements, Capabilities, Policies types definition of ECM DSL that define specific components of the language that can be instantiated in template.

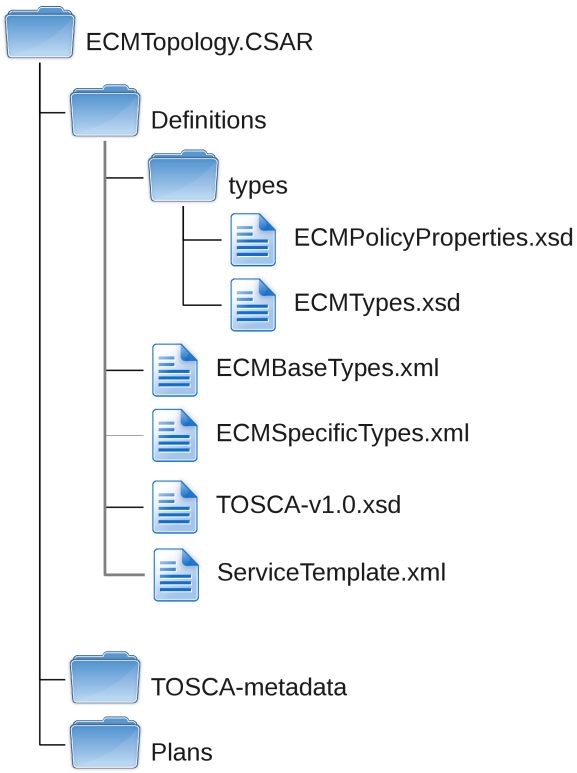

Figure 4.18: ECM DSL CSAR archive structure

- "TOSCA-metadata" folder contains information about CSAR file that is used by execution environment.
- "Plans" folder contains deployment plans.
- "ServiceTemplate.xml" contains Service template, where Policy, Node, Relationship templates are defined.

DSL execution The TOSCA standard defines execution environment (TOSCA container) that can process CSAR archive and deploy service instance. It requires definition of the artifacts, providing component binaries, etc. ECM DSL defines ECM components using high level abstraction. It means that nodes are independent from the particular solution. Additionally, ECM language defines node types that are used only for definition of non-functional characteristics and should not be deployed in the ECM solution (e.g. Content Node, Business Node, IT Node). These non-functional terms allow to define requirements of ECMS. For example, definition of "Mail Server" and "File Server" and declaration of "ContentFlow" relationships from these components to "Content Collector" node declares that the content should be collected from mail and file servers and should be stored in the ECM repository. It defines additional requirements for content collector for mail and file server protocols support. According to this description, TOSCA execution environment can not be used, because it

does not allow to make an analysis of the ECM topology (calculation of average documents size, definition of enterprise storage resources, etc.).
An additional adapter, that will be able to analyze ECM topology and translate high level terms to particular ECM components, can be developed by cloud providers.

5 Evaluation

The development of the ECM language was done in an iterative way. The concepts of the language were discussed with IBM developers, and changes were made according to their feedback. For example, on the early stage of the DSL development, quantitative parameters of ECMS was provided as properties of the ECM classes. After the presenting of the language solution, it was decided to declare these parameters as a part of a policy class, because they reflect properties of a corporate policy.
The evaluation of the DSL was done by implementing the use case provided by IBM. Then the example of the defined topology was presented to the IBM ECM developers and the feedback was received by providing the questionnaire that was defined during the domain analysis phase. The modeling was done using Winery tool.
In the use case records manager of the enterprise describes records management requirements using a language that he/she understands: "As the Record Manager of my enterprise I want to define the corporate taxonomy being able to classify documents and business relevant information into mandate corporate record categories."
ECM architect defines ECM related terms and actions that should be supported by ECMS. From the following example the architect defines the following terms: "enterprise", "taxonomy", "document", "business information", "record category". The architect also defines the following actions: "define", "classify", "mandate". After the definition of the vocabulary, ECM architect models a template using ECM language by choosing nodes, that represent the ECM vocabulary. He chooses "Records Management" that supports operations defined by the Record Manager. If the node supports not all operations, that were defined by Record Manager, then ECM architect creates new node, that inherits properties of the already existed node, and adds operations that should be supported by the node. In the current example, the "Records Management" node should be extended if it supports not all operations defined by Records Manager. Then ECM architect adds additional nodes that should be defined for the topology completeness. In this case the following nodes should be added to the template: "ECM Repository", "File Plan", "Retention Schedule". Additionally, ECM Architect can define ECM Access nodes that will describe, how a user will interact with ECMS. He/she also can specify execution environment nodes and the policy that should be applied to ECMS. Then business customer fills policy and topology properties. The result of the ECM service template modeling is shown on Figure 5.1.
After the description of the ECM language and presentation of the use case, the following questions were asked:

- Is the presented language useful for definition of quantitative and qualitative characteristics and policies of ECM System?

- Are the ECM components described using high level abstraction and are they independent from particular components from different vendors (IBM, Alfresco, Oracle, etc.)?
- Does the presented ECM template allow to define components that should be deployed?
- Is it possible to define Quantitative characteristics (maximum number of concurrent users, average size of files, number of file shares, etc.) from provided ECM service template?

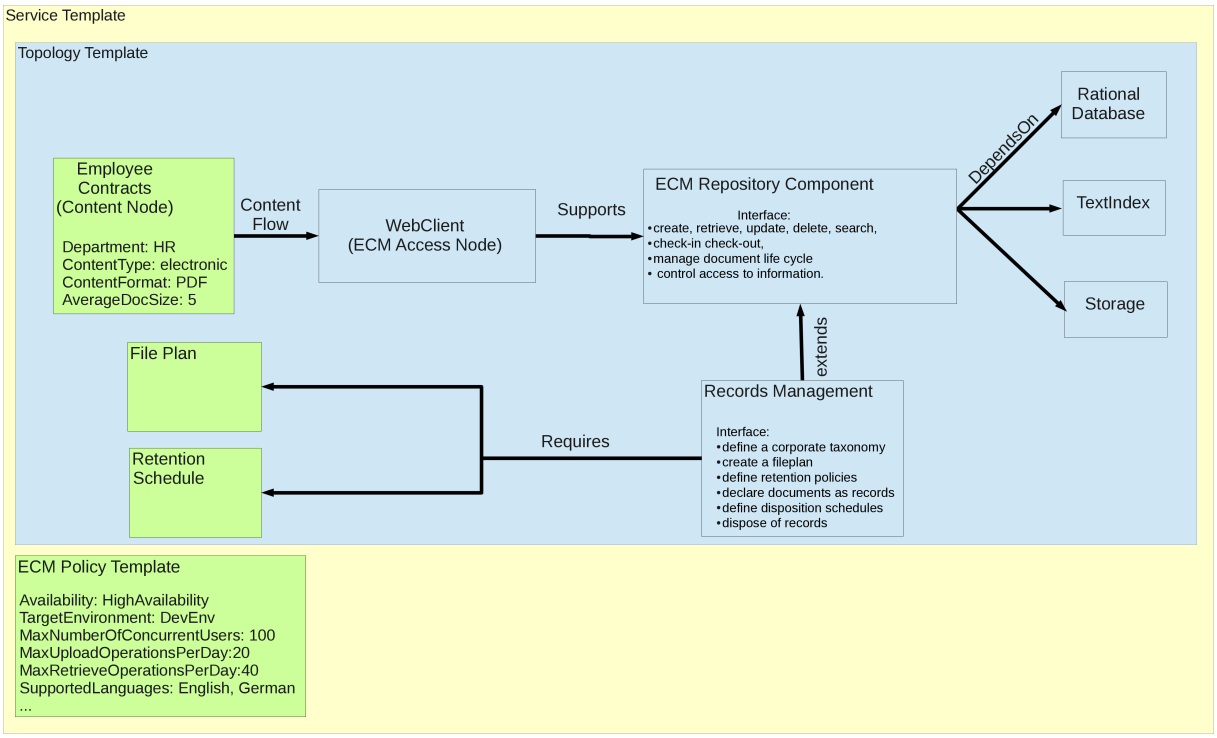

Figure 5.1: ECM records management service template

The results of the evaluation show that the presented language can be useful for solutions following IBM's architecture patterns for ECM. The components from different vendors can be subsumed under one or other term of the ontology, but the discussion with other ECM vendors is required. The attributes that were provided by the language template allow to define quantitative requirements. Additionally, was stated that the usage of TOSCA for the concrete syntax definition provides limitations to the developed language. The usage of the TOSCA language does not allow to use XML parser for the model validation. Additional tools (e.g. Winery) should used for the modeling and validation purposes.

6 Conclusion

In this thesis I developed DSL for ECM domain. The result of the work is ECM ontology and the TOSCA based declarative language that allows ECM architect to define functional and non-functional requirements provided by the customer. The definition of the requirements can be done by creating a service template, that consists of the ECM terms, their definitions, and the set of operations that are associated with each term. The analysis of the commonalities and variabilities of the ECM solutions from different vendors showed that the workloads with the same names can provide different meanings. ECM DSL solves this problem by normalizing ECM related terms and providing operations that are associated with each term. The applied domain analysis method allows to use the developed ontology as a basis for the future research of ECM domain. It was designed in a such way that new classes can be added as children of the existed classes without changing structure of the ontology. Introduction of content node allows to define all types of the content captured by enterprise. Enterprise components and content flow relationships allows to describe business and IT components of enterprise and show how the information is categorized and stored within enterprise. ECM nodes and relationships between them allow to design the ECM topology and define workloads of ECM components.
TOSCA modeling tools (e.g Winery) and custom adapters can be used for designing, analysis and deployment of the ECM service template. The exploitation of the TOSCA language allows to define ECM related terms, their properties and associated operations. The node types represent ECM and Enterprise terms in high abstraction level. It means that the node types do not depend on the particular ECM solution, and can be used by different ECM developers and architects. For each ECM related term the TOSCA node type and the interface, that describes supported operations was created . Appliance of the TOSCA policies to ECM allows to define required KPIs and provide deployment restrictions. Cloud providers can analyze the resulted graph represented by the TOSCA topology and derive qualitative and quantitative information from it.
Usage of the TOSCA language was limited by the definition of node, requirement and capability, relationship and policy types and templates. TOSCA was designed for the deployment automation of cloud services and most of its syntax was not used during ECM language development. Usage of TOSCA language was provided as a part of the thesis topic.
The evaluation of the model was done by creating the ECM service template based on the typical ECM requirements provided by IBM and presenting them to ECMS developers. The feedback was received by providing questionnaire to ECM developers. The feedback showed that the developed language can be used for solutions following IBM's architecture patterns for ECM. It is possible to define qualitative and quantitative parameters from the provided service template. Additionally, it was noticed that the usage of the TOSCA language limits the application of DSL because it requires too much custom logic to verify different ECM

solutions. This limitation can be overcome by extending TOSCA XML schema and creating ECM elements that extends TOSCA node types defined in schema. It allows to use XML parsers for the validation of the created templates.

7 Appendix A: ECM ontology

Class	**Superclass**	**Description**	Properties		
			Name	Description	Value
Content	RootNode	Information about the content in enterprise	Department	Name of the department that acquires the content	String
			ContentType	Type of the acquired content	Enumerated: Paper |Electronic
			ContentFormat	Format of the electronic captured component	Enumerated: PDF |JPEG |XML
			AverageDocumentSizeMB	Average size of the document of the type in Megabytes	Integer
			DocumentCategory	Document category for the current content object	Class TaxonomyCategory
Enterprise	-	A Generic class that unites subclasses that describes Business and IT structure in Enterprise	Name	Name of the department that acquires the content	-
Enterprise Information Structure	Enterprise	Describes Enterprise Business Structure	-	-	-
Enterprise Taxonomy	Enterprise Information Structure	Classification Scheme of Enterprise Information	TaxonomyCategory	Document Classes that are used for information classification	Class TaxonomyCategory

Continued on next page

Class	**Superclass**	**Description**	Properties		
			Name	Description	Value
RetentionSchedule	Enterprise Information Structure	Contains rules for retention periods and dispositions methods of enterprise records	Retention Rule	Rule that defines the period for records preservation, actions that initiate that period and disposition action action after that period	Class RetentionRule
FilePlan	Enterprise Information Structure	The hierarchical structure of enterprise records	ClassificationScheme	Records classification Scheme in Enterprise	Enumerated: Functional Subject Organizational
			RecordItem	Function, Activity or Records Series that describes Records structure in Enterprise	Class RecordItem
EnterpriseIT	Enterprise Component	Enterprise IT Infrastructure that is used for Capture, Store, and Deliver Enterprise content	location	Geographical location of the component	String
Enterprise Capture	EnterpriseIT	Enterprise Hardware or software component that creates or captures Enterprise information	Manufacturer	The manufacturer of enterprise Component	String
			Version	The model of Hardware component or version of Software component	String
			Documents CapturePer Day	Average number of documents that are captured/created by the component	Integer
EnterpriseStorage	EnterpriseIT	Software or Hardware component that is used for storage of Enterprise content	-	-	-

Continued on next page

Class	**Superclass**	**Description**	Properties		
			Name	Description	Value
Enterprise Delivery	EnterpriseITComponent	Enterprise Software or hardware component for content delivery	-	-	-
Disposal	Enterprise	Describes the disposition methods of the content at the end of the content's lifecycle	-	-	-
ECM	-	Root node of ECM terms hierarchy	Name	Name of the instance of the class	String
			Interface	Define workloads and associated operations that should be supported by the node	Class ECMInterface
ECMAccess	ECM	Term that describes capture and delivery of enterprise information			
Imaging	ECMAccess	Software Component provided by the ECM Provider/Developer for scanning, recognition, Image Enhancement	Recognition Technologies	List of Recognition technologies that should be supported by the imaging component: OCR - typed text recognition HCR - handwritten text recognition barcode - recognition of the barcodes	Enumerated
			ScanningFormats	List of the formats that should be supported by the Imaging component	Enumerated
WebClient	ECMAccess	Web based component for accessing to the ECM Management Operations	-	-	-

Continued on next page

Class	**Superclass**	**Description**	Properties		
			Name	Description	Value
API	ECMAccess	Application Programming Interface that allows to third party software to access ECM Management component functions	APIType	List of the API type that should be supprted by ECM Manage component: NATIVE - allows to call functions using native ECMS language CMIS - operations that are supported by Content Management Interoperability Standard WebServices - envoke ECMS commands using web services	Enumerated
Content Collector	ECMAccess	Component for Collecting Content from Enterprise storage (Servers, Mail Servers, Unstructured reposiroties, etc), categorization this content according to enterprise taxonomy, and retrieving the index from them	-	-	-
ECMmanage	ECM	Describe terms that are associated with a management of enterprise content	-	-	-
ECMFoundation	ECMManage	Foundation of ECM solution	-	-	-
ECMRepository	ECMFoundation	Library services (create, retrieve, update, delete, search, check-in, check-out, etc.) and a storage for content and its metadata	ExtendedBy	Components that extends basic functionality of Repository	Class ECM ExtensionComponent

Continued on next page

Class	**Superclass**	**Description**	Properties		
			Name	Description	Value
ECMExtention	ECMNode	Generic class that adds additional functionality to ECM foundation	-	-	-
RecordsManagement	ECMExtension	Manages Enterprise Records according to the Enterprise File Plan and Retention Schedule	Requires	Components that should be defined for the corrected execution of Records Management component	Class RecordComponent
Ediscovery	ECMExtention	requirements for a process of discovery in civil litigation that is carried out in electronic format	-	-	-
Search	ECMExtention	Customer Requirements for full text and parametric search	-	-	-
ECMGovern	ECM	This term enables creation of Retention Policy and Schedule Management, Ediscovery Process Management, Disposal and Governance Management for IT	-	-	-
TaxonomyCategory	-	Enterprise taxonomy's hierarchical component	derivedFrom	Declares the id of the derived taxonomy category	String
			ID	ID of the instance	String
FilePlanCategory	-	Describes RM Function or Activity	derivedFrom	Declares the id of the derived file plan category	String
			ID	ID of the instance	String
			Description	Description of the Record item	String

Continued on next page

Class	**Superclass**	**Description**	Properties		
			Name	Description	Value
			AccessRoles	Link to the roles that have the access to records	Class AccessRole
RecordsSeries	FilePlanCategor	Records that are related to the particular subject or function	RecordFormat	Format of the record	Enumerated: Paper, Electronic
		|	ArrangedBy	Type of records arrangement	Enumerated: Date, Surname, DocumentNumber
			DateOpened	Open Date of the record	Date
			DateCreated	Creation Date of the record	Date
			DateClosed	Close Date of the record	Date
Retention Rule	-	Rule that defines the period for records preservation, actions that initiate that period and disposition action action after that period	Description	Description of the retention Rule	String
			ID	ID of the instance	String
			RetentionPeriodMonths	Retention period for record preservation, after which a record wil be disposed. If the record is onHold that the record will not be disposed after retention period	Integer

Continued on next page

Class	**Superclass**	**Description**	Properties		
			Name	Description	Value
			DispositionAction	Disposition action that should be executed after the retention period: Archive - save in long term archive Shred - destruction of the record	Enumerated
			StartTrigger	An action after which retention period is started	Enumerated
Policy	-	A definition of the policy that should applied to ECM solution	Name	Name of the instance of the class	String
			SupportedLanguages	List of supported languages by ECM component	Enumerated
			MaxNumberOfConcurrentUsers	Maximum number of ECM users that work with the component at the same time	Integer
			MaximumUploadsPerDay	Maximum number of content objects that are uploaded using current component per day	Integer
			MaximumRetrievsPerDay	Maximum number of content objects that are retrieved using current component per day	Integer
			MaximumSearchesPerDay	Maximum number of search requests done using a component per day	Integer
			NumberOfDocumentsLoadedPerTime	Number of content objects that are uploaded during batch loading	Integer

Continued on next page

Class	Superclass	Description	Properties		
			Name	Description	Value
			Supports	Components that are supported by the Web Client	Class ECMExtensionComponent, ECMCoreComponent
			SupportedBrowsers	List of the supported web browsers	Enumerated
Relationship	-	Generic class that describes relationships between language terms	Name	Name of the instance of the class	String
capturedBy	Relationship	Relationship describes which content should be collected using ECM Access method	source	Source defines from which class the relationship is established	Class "Content"
			target	Target class	Class "ECM Access"
exportedTo	Relationship	Relationship defines to which formats the content should be transformed	source	Source defines from which class the relationship is established	Class "ECMAccess"
			target	Target class	Class "Content"
flows	Relationship	The relationship describes the content flow within enterprise.	source	Source defines from which class the relationship is established	Class "Content"
			target	Target class	Class "EnterpriseIT"

Continued on next page

Class	**Superclass**	**Description**	Properties		
			Name	Description	Value
collectedBy	Relationship	The relationship defines from which enterprise storage resources the data should be collected and stored in ECM repository	source	Source defines from which class the relationship is established	Class "EnterpriseIT"
			target	Target class	Class "ECM Access"
supports	Relationship	the relationship define which ECM Manage interfaces should be associated with ECM Access term.	source	Source defines from which class the relationship is established	Class "ECM Access"
			target	Target declares a destination of the relationship	Class "ECMManage"
appliedTo	Relationship	The relationship defines the link between policy and ECM terms. It declares to which ECM nodes the defined policy should be applied	source	Source defines from which class the relationship is established	Class "Policy"
			target	Target declares a destination of the relationship	Class "ECM"
dependsOn	Relationship	The relationship define dependency between ECM and Execution environment nodes.	source	Source defines from which class the relationship is established	Class "ECM"
			target	Target declares a destination of the relationship	Class "ExecutionEnvironment"

Continued on next page

Class	**Superclass**	**Description**	Properties		
			Name	Description	Value
extends	Relationship	The relationship define which ECM extentions adds functionality to ECM foundation. It also declares that ECM extention term depends on ECM Foundation term	source	Source defines from which class the relationship is established	Class “ECMExtension”
			target	Target declares a destination of the relationship	Class “ECMFoundation”
requires	Relationship	The Relationship defines dependencies between ”ECM extension” and ”Enterprise” classes”. It declares that an instance of ”ECM Extension” requires information about an enterprise	source	Source defines from which class the relationship is established	Class “ECMExtension”
			target	Target declares a destination of the relationship	Class “Enterprise”

8 Appendix B: ECM foundation service template using TOSCA

```
<Definitions id="ECMFoundation"
        targetNamespace="http://docs.oasis-open.org/tosca/ns/2011/12"
        xmlns="http://docs.oasis-open.org/tosca/ns/2011/12"
        xmlns:ns1="URI"
        xmlns:ns2="URI"
        xmlns:xsi="URI">

<Import importType="http://www.w3.org/2001/XMLSchema"
        location="Definitions/types/ECMTypes.xsd"
        namespace="http://boeblingen.de.ibm.com/tosca/types/ECM" />
<Import importType="http://www.w3.org/2001/XMLSchema"
        location="Definitions/ECMSpecificTypes.xml"
        namespace="http://boeblingen.de.ibm.com/tosca/
        types/ECMSpecificTypes" />

<!-- Policy Template -->
<PolicyTemplate type="ns1:ECMPolicy" id="ECMPolicyTemplate">
        <Properties>
                <ns2:ECMPolicyProperties>
                        <Availability>HighAvailability</Availability>
                        <TargetEnvironment>DevEnv</TargetEnvironment>
                        <MaxNumberOfConcurrentUsers>10
                        </MaxNumberOfConcurrentUsers>
                        <MaxUploadsPerDay>20</MaxUploadsPerDay>
                        <MaxRetrievesPerDay>40</MaxRetrievesPerDay>
                        <MaximumSearchesPerDay>55
                        </MaximumSearchesPerDay>
                        <NumberOfDocumentsLoadedPerTime>90
                        </NumberOfDocumentsLoadedPerTime>
                        <SupportedLanguages>
                                <Language>English</Language>
                                <Language>German</Language>
                        </SupportedLanguages>
                </ns2:ECMPolicyProperties>
        </Properties>
```

```
</PolicyTemplate>

<ServiceTemplate id="ECMFoundation" name="ECMFoundation"
        targetNamespace="http://docs.oasis-open.org/tosca/ns/2011/12">
<BoundaryDefinitions>
        <Interfaces>
                <Interface name="RepositoryUserInterface">
                <Operation name="createDocument">
                        <NodeOperation operationName="createDocument"
                                nodeRef="ECMRepositoryNode"
                                interfaceName="UserInterface" />
                </Operation>
                <Operation name="retrieveDocument">
                        <NodeOperation operationName="retrieveDocument"
                                nodeRef="ECMRepositoryNode"
                                interfaceName="UserInterface" />
                </Operation>
                <Operation name="updateDocument">
                        <NodeOperation operationName="updateDocument"
                                nodeRef="ECMRepositoryNode"
                                interfaceName="UserInterface" />
                </Operation>
                <Operation name="deleteDocument">
                        <NodeOperation operationName="deleteDocument"
                                nodeRef="ECMRepositoryNode"
                                interfaceName="UserInterface" />
                </Operation>
                <Operation name="searchDocument">
                        <NodeOperation operationName="searchDocument"
                                nodeRef="ECMRepositoryNode"
                                interfaceName="UserInterface" />
                </Operation>
                <Operation name="checkIn">
                        <NodeOperation operationName="checkIn"
                                nodeRef="ECMRepositoryNode"
                                interfaceName="UserInterface" />
                </Operation>
                <Operation name="checkOut">
                        <NodeOperation operationName="checkOut"
                                nodeRef="ECMRepositoryNode"
                                interfaceName="UserInterface" />
                </Operation>
                </Interface>
        </Interfaces>
</BoundaryDefinitions>
```

```
<TopologyTemplate>
<!-- Enterprise Nodes -->
        <NodeTemplate name="Employee Contract" id="ContentNode"
                type="ns1:ContentNode">
                <Properties>
                        <ns2:ContentNodeProperties>
                                <Department>HR</Department>
                                <ContentType>Electronic</ContentType>
                                <ContentFormat>PDF</ContentFormat>
                                <AverageDocSize>5</AverageDocSize>
                        </ns2:ContentNodeProperties>
                </Properties>
        </NodeTemplate>

<!-- ECM Nodes -->
        <NodeTemplate name="WebClient" id="WebClient"
                type="ns1:WebClient">
                <Policies>
                        <Policy policyType="ns1:ECMPolicy"
                        policyRef="ECMPolicyTemplate" />
                </Policies>
        </NodeTemplate>

        <NodeTemplate name="Repository" id="ECMRepositoryNode"
                type="ns1:ECMRepositoryNode">
                <Requirements>
                        <Requirement name="DataBaseRequirement"
                                type="ns1:RationalDataBaseRequirement"
                                id="DBRequirement" />
                        <Requirement name="StorageRequirement"
                                type="ns1:StorageRequirement"
                                id="StorageRequirement" />
                        <Requirement name="TextIndexRequirement"
                                type="ns1:TextIndexRequirement"
                                id="TextIndexRequirement" />
                </Requirements>
                <Capabilities>
                        <Capability name="ECMAddOnCapability"
                                type="ns1:AddonContainerCapability"
                                id="AddOnCapability"/>
                </Capabilities>
                <Policies>
                        <Policy policyType="ns1:ECMPolicy"
                                policyRef="ECMPolicyTemplate" />
                </Policies>
```

```
        </NodeTemplate>

<!-- Execution Environment Node -->
        <NodeTemplate name="TextIndex" id="TextIndex"
                type="ns1:TextIndex">
                <Capabilities>
                        <Capability name="TextIndexCapability"
                                type="ns1:TextIndexCapability"
                                id="textIndexCpability" />
                </Capabilities>
        </NodeTemplate>

        <NodeTemplate name="Storage" id="Storage"
                type="ns1:Storage">
                <Capabilities>
                        <Capability name="StorageCapability"
                                type="ns1:StorageCapability"
                                id="StorageCapability"/>
                </Capabilities>
        </NodeTemplate>

        <NodeTemplate name="RationalDatabase"
                id="RationalDatabase"
                type="ns1:RationalDatabase">
                <Capabilities>
                        <Capability name="DBCapability"
                        type="ns1:RationalDataBaseCapability"
                        id="RDBCapability"/>
                </Capabilities>
        </NodeTemplate>

<!-- Relationships Between Nodes -->
        <RelationshipTemplate
                name="ContentNode_WebClient_ContentFlow"
                id="contentflow1" type="ns1:ContentFlow">
                <SourceElement ref="ContentNode" />
                <TargetElement ref="WebClient" />
        </RelationshipTemplate>

        <RelationshipTemplate
                name="WebClient_supports_Repository"
                id="supports1" type="ns1:Supports">
                <SourceElement ref="WebClient" />
                <TargetElement ref="ECMRepositoryNode" />
        </RelationshipTemplate>
```

```
        <RelationshipTemplate
                name="Repository_dependsOn_TextIndex"
                id="dependson1" type="ns1:DependsOn">
                <SourceElement ref="ECMRepositoryNode" />
                <TargetElement ref="TextIndex" />
        </RelationshipTemplate>

        <RelationshipTemplate
                name="Repository_DependsOn_Storage"
                id="dependson2" type="ns1:DependsOn">
                <SourceElement ref="ECMRepositoryNode" />
                <TargetElement ref="Storage" />
        </RelationshipTemplate>

        <RelationshipTemplate
                name="Repository_DependsOn_DB"
                id="dependson3" type="ns1:DependsOn">
                <SourceElement ref="ECMRepositoryNode" />
                <TargetElement ref="RationalDatabase" />
        </RelationshipTemplate>
</TopologyTemplate>
</ServiceTemplate>
</Definitions>
```

Bibliography

[AII14] What is enterprise content management (ecm)? http://www.aiim.org/What-is-ECM-Enterprise-Content-Management, 2014.

[BB07] Joseph A. Busch and Lisa Butcher. Getting started with business taxonomy design. http://www.taxonomystrategies.com/presentations/2010/Business2007. Accessed: 2015-04-20.

[BBFRS12] Cynthia Beth, Irma Becerra-Fernandez, Jeanne Ross, and James Short. Finding values in the information explosion. *MIT Sloan Management Review*, 53(4), 2012.

[Bö14] Martin Böhn. The market of ecm software. *Enterprise Content Management in Information Systems Research*, pages 23–36, 2014.

[Cam11] Stephen Cameron. *Enterprise Content Management: A business and Technical Guide*. 2011.

[EMC08] Emc documentum architecture: Delivering the foundations and services for managing content across the enterprise, 2008.

[GHH+06] Knut R. Grahlmann, Remko Helms, Cokky Hilhorst, Sjaak Brinkkemper, and Sander van Amerongen. Reviewing enterprise content management: A functional framework. 2006.

[Gho11] Debasish Ghosh. Dsls in action. Manning Pupblications Co., 2011.

[GSC+14] Mark R. Gilbert, Karen M. Shegda, Kenneth Chin, Gavin Tay, and Hanns Koehler-Kruener. Magic quadrant for enterprise content management, 2014.

[HEI07] Hei records management. guidance on developing a file plan. http://tools.jiscinfonet.ac.uk/downloads/bcs-rrs/developing-a-file-plan.pdf, 2007. Accessed: 2015-04-05.

[idg14] Investments and upgrades in cloud solutions drive business agility and innovation. http://www.idgenterprise.com/press/investments-and-upgrades-in-cloud-solutions-drives-business-agility-and-innovation/, 2014.

[Jen04] Tom Jenkins. *Enterprise Content Management*. Open Text Corporation, 2004.

[KKP+14] Gabor Karsai, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler, and Steven Völker. Design guidelines for domain specific languages. 2014.

[Kum10] Pawan Kumar. *Documentum 6.5 Content Management Foundations*. 2010.

[LBMCW13] Dr. Sven Laumer, PD Dr. Daniel Beimborn, Dipl.-Wirtsch.Inf. Christian Maier, and M.Sc. Christoph Weiner. Enterprise content management. *WIRTSCHAFTSINFORMATIK*, 55(6), 2013.

[MG11] Peter Mell and Tymothy Grance. The nist definition of cloud computing. recommendations of the national institute of standards and technology. *National Institute of Standards and Technology Special Publication 800-145*, 2011.

[MHS05] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4):316–344, 2005.

[NM] Natalya F. Noy and Deborah L. McGuiness. Ontology development 101: A guide to creating your first ontology.

[oT14] California Department of Technology. Enterprise content management (ecm). reference architecture. 2014.

[PeM05] Tero Päivärinta and Bjorn erik Munkvold. Enterprise content management: An intergated perspective on information management. *Proceedings of the 38th Hawaii International Conference on System Sciences*, 2005.

[Sma14] Robert F. Smallwood. *Information Governance for business documents and records*. 2014.

[Tos13a] *Topology and Orchestration Specification for Cloud Applications Primer Version 1.0*. 2013.

[Tos13b] *Topology and Orchestration Specification for Cloud Applications Version 1.0, OASIS Standard*. 2013.

[ZAB+09] Wei-Dong Zhu, Richard Aitchison, Eric Bonner, Hector Casals Mendez, Ron Rathgeber, Amit Yadat, and Harry Yessayan. *Understanding IBM FileNet Records Manager*. 2009.

[ZBO+11] Wei-Dong Zhu, Nicholas Buchanan, Michael Oland, Thorsten Poggensee, Pablo E Romero, Chuck Snow, and Margaret Worel. *IBM FileNet P8 Platform and Architeture*. 2011.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature