D4.2: Dependency Models Iter. 1:
Definition of a formalism to express dependencies and relations between technological, business and organizational components and processes.

WP 4 – Processes and Methods for Digitally Preserving Business Processes

Delivery Date: 30/03/2012

Dissemination Level: Restricted

Nature: Report

TIMBUS is supported by the European Union under the 7th Framework Programme for research and technological development and demonstration activities (FP7/2007-2013) under grant agreement no. 269940

Deliverable Lead		
Name	**Organisation**	**e-mail**
John Thomson	Caixa Mágica Software	john.thomson@caixamagica.pt

Contributors		
Name	**Organisation**	**e-mail**
Paulo Trezentos	Caixa Mágica Software	paulo.trezentos@caixamagica.pt
Mário Romão	Caixa Mágica Software	mario.romao@caixamagica.pt
Ricardo Teixeira	Caixa Mágica Software	ricardo.teixeira@caixamagica.pt
Andreia Palma	Caixa Mágica Software	andreia.palma@caixamagica.pt
Hedda Schmidtke	KIT	schmidtke@teco.edu
Martin Alexander Neumann	KIT	mneumann@teco.edu
Gonçalo Antunes	INESC-ID	goncalo.antunes@ist.utl.pt
Diogo Proença	INESC-ID	diogobcp@gmail.com
Artur Caetano	INESC-ID	artur.caetano@ist.utl.pt
Mike Nolan	Intel	michael.nolan@intel.com
Daniel Draws	SQS	daniel.draws@sqs.com
Gregor Heinrich	iPharro	g.heinrich@ipharro.com
Rudolf Mayer	SBA	mayer@ifs.tuwien.ac.at
David Redlich	SAP	david.redlich@sap.com

Internal Reviewer		
Name	**Organisation**	**e-mail**
PCC: William Kilbride	DPC	william@dpconline.org
Peer: Stephan Strodl	SBA	sstrodl@sba-research.org

Disclaimer

use of these materials subject to any liability which is mandatory due to applicable law.

This project has been funded with support from the European Commission.
This publication, Deliverable 4.2, reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

TABLE OF CONTENTS

List of Figures

List of Tables

List of Acronyms

WP{X}	Work Package X of TIMBUS
D{X.Y}	Deliverable X.Y of TIMBUS, where X is the Work Package
T{X.Y}	Task X.Y of TIMBUS, where X is the Work Package and Y specific task
CMS	Caixa Mágica Software
DPC	DIGITAL PRESERVATION COALITION LIMITED BY GUARANTEE
INESC-ID	INSTITUTO DE ENGENHARIA DE SISTEMAS E COMPUTADORES, INVESTIGACAO E DESENVOLVIMENTO EM LISBOA
KIT	Karlsruher Institut fuer Technologie
SAP	SAP AG
SBA	VEREIN ZUR FORDERUNG DER IT-SICHERHEIT IN OSTERREICH
SQS	SQS SOFTWARE QUALITY SYSTEMS AG
ACL	Agent Communication Language
ADF	Activity-Decision Flow
AIDC	Automatic Identification and Data Capture
API	Application Programming Interface
BAT	Big Air Transport
BPD	Business Process Diagram
BPEL4WS	Business Process Exececution Language for Web Services
BPMI	Business Process Modelling Initiative
BPM	Business Process Management
BPMN	Business Process Model Notation
BPMS	Business Process Management Systems
BSDL	Business Service Description Language
CASPAR	Cultural, Artistic and Scientific knowledge for Preservation, Access and Retrieval (http://www.casparpreserves.eu/)
CD	Compact Disc
CM	Configuration Management
CMDB	Configuration Management Database
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CMU	Carnegie Mellon University
COBIT	Stood for, Control Objectives for Information and related Technology
CUDF	Common Upgrade Descripition Format (www.mancoosi.org/cudf)
DL	Description Logic
DLL	(Windows) Dynamically Linked Library

DPE	DigitalPreservationEurope (http://www.digitalpreservationeurope.eu/)
DSL	Domain Specific Language
DVD	Digital Versatile Disc
EI	Enterprise Integration
ETL	Extraction, Transform and Load
FIPA	Foundation for Intelligent Physical Agents
GPL	Context of programming/modelling or languages - General Purpose Languages
HCL	Hardware Compatibility List
HTTP	Hyper-text transfer protocol
HTTPS	Hyper-text transfer protocol Secure
HW	Hardware
iAMT	Intel Active Management Technology
ICMP	Internet Control Message Protocol
IDEF	Integration Definition
IEC	International Engineering Consortium
IEEE	Institute of Electrical and Electronics Engineers
IERM	Intelligent Enterprise Risk Management (specified in D4.1 – TIMBUS)
IoS	Internet of Services
IP	Internet Protocol
ISO	International Organisation for Standardisation
IT	Information Technology
ITIL	Information Technology Infrastructure Library
JADE	Java Agent Development Framework
KQML	Knowledge Query and Manipulation language
NIST	National Institute of Standards and Technology
NT	(Windows) New Technology
OA	Organisational Agents
OAIS	Open Archival Information System
OASIS	Organisation for the Advancement of Structured Information Standards
OGC	Government Commerce
OMG	Object Management Group
OS	Operating System
OWL	Web Ontology Language
PaaS	Platform as a Service
PREMIS	PREservation Metadata: Implementation Strategies working group
PXE	Pre eXecution Environment
RDF	Resource Description Framework
REST	Representational state transfer

RMM	(Intel) Remote Management Module
RPM	RPM Package Manager
RUP	Rational Unified Process
SaaS	Software as a Service
SAT	Satisfiability Problem
SCM	Software Configuration Management
SEI	Software Engineering Institute
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
SoaML	Service oriented architecture Modelling Language
SOAP	Used to stand for Simple Object Access Protocol
SSQC	Software Systems Quality Consulting
SW	Software
SWEBOK	Software Engineering Body of Knowledge
TOGAF	The Open Group Architecture Framework
TOVE	Toronto Virtual Enterprise project
UML	Unified Modelling Language
URL	Uniform Resource Locator
VM	Virtual Machine
VSRI	Virtualisation, Storage, Rerun and Integration
WMI	Windows Management Instrumentation
WS-BPEL	Web Services Business Process Execution Language
WSDL	Web Service Description Language
WSH	Windows Script Host
XML	Extensible Markup Language
XPDL	XML Process Definition Language
XSD	XML Schema Definition

1 Executive Summary

Digital preservation is traditionally understood as the management of digital information for as long as necessary [Beagrie N., et al., 2001]. It is the set of processes and activities that ensure continued access to assets existing in digital formats.

The TIMBUS Project looks to enlarge the understanding of digital preservation to include the set of activities, processes and tools that ensure continued access to services and software necessary to produce the context within which information can be accessed, properly rendered, validated and transformed into knowledge. One of the fundamental requirements is to preserve the functional and non-functional specifications of services and software, along with their dependencies. Service dependency analysis is fundamental in determining what should be preserved. For that purpose, TIMBUS will use a combination of both manual and intelligent systems that can be used to integrate the results from enterprise risk management, service dependency analysis and value engineering.

Work Package 4 of TIMBUS looks to investigate what is required for digital preservation to be performed in an Enterprise System. As such, the analysis of the related work and research will be carried out within Work Package 4 and then in Work Package 5, a specific architecture proposed that will be implemented through tools in Work Package 6. The tools that are implemented as a result of Task 4.2 and linked to this deliverable are D6.2, Dependencies Monitor & Reasoning System and D6.5, Populating and Accessing Context Model.

The aim of Task 4.2 is to develop a means for describing the dependencies between different components of an Enterprise Process through the different layers of an Enterprise. To identify the types of dependencies required it is essential to categorise the types of layers in an Enterprise and determine the components that are needed for preserving a business process.

Given the large diversity of Enterprises, the overall approach to dependency capturing has to be generic enough for encompassing such diversity. In this deliverable a first version of Formalism for dependency modelling is proposed. It is design to be extensible in order to accommodate such diversity, with this first iteration focusing mainly on the technical aspects surrounding a business process. It is based on the analysis of relevant related work so that it is in line with established Domain Specific Languages (DSLs). Two types of dependency relationships were derived: constraint relations, which are strict relations that must be met in order in order to effectively preserve a process; and description relations, which associate entities with attribute details. This deliverable also depicts the application of the developed formalism to a use case dealing with a Music Classification Process. The purpose is to show that the

main result of this work has enough expressiveness to capture the properties of the technical infrastructure supporting that process.

The formalism was developed jointly with Task 4.4 under the scope of D4.5, which focus on context modelling. The result was a unified model which is able to capture dependencies on the context surrounding a business process. While the present deliverable focuses on the dependency relationships, D4.5 focuses on the description of relevant context parameters.

2 Introduction

In order to be rendered, digital objects depend on a technological context defined by specific combination of software and hardware that is able to decode the bit organisation. In the traditional approaches to digital preservation, the focus is on the preservation of the digital object itself along with extra information required to be able to render the object in the future, usually known as technical metadata, which can contain information about the technical environment involved in the production or usage of an object.

However, that alone is not sufficient to be able to make sense of the informational content carried by the object, since it requires a social or organisational context where it was created or used. This fact has been highlighted in the digital preservation Europe research roadmap [DPE 2007], which defines the context of a digital object as the “representation of known properties associated with and the operations that have been carried out on it”. Those properties might include information about technology, legal requirements, existing knowledge, and user requirements. Thus, the reuse of a digital object might depend of any of those factors.

The challenges associated with this issue are even more complex if we consider complex digital objects such as business process or workflow specifications that are dependent on highly distributed service environments supported by heterogeneous technologies that are running in highly diverse organisational settings. The effective preservation and authoritative re-enactment of such objects might involve the capturing of other digital objects that are also dependent on other objects, forming a complex network of dependencies.

In order to tackle this challenge, the TIMBUS project has the overall goal of enabling the successful digital preservation of business processes, which is an innovative concept in the digital preservation community. For that the entire relevant context has to be captured using automatic or semi-automatic means, so that the process can be exhumed and re-enacted. This will necessarily involve the modelling of all the possible dependencies existing between digital objects, so that all the required data can be tracked and preserved.

2.1 Dependencies

Dependencies as described in this deliverable in general are a descriptive term for relationships between two entities. This largely follows the definition of dependencies and coupling that is used in computer science [Stevens, W.P, et al., 1974]. In abstract terms, if entities are represented as nodes in a graph, then dependencies are the edges between these nodes. The type of relationship as denoted by the edge is

determined by what the graph is representing though. There may be many similar types of relationships that can be linked in this way.

In general:

Entity_1	Relationship	Entity_2

Where **Entity1** and **Entity2** are two different entities or different specific instantiations of entities and **Relationship** denotes the type of relationship the entities have on each other. These relationships might be directed, so the order used for specifying a dependency is important.

The definition of a dependency is actually more specific than the general entity relationships and it denotes a requirement that one entity must be present at the same time as the other. An entity that is dependent on another is known as the dependent and the entity to which it refers can be called the dependee.

The concept of dependency can be applied both to tangible or intangible things. For instance, the enactment of a business process might be dependent on a determined business actor, or of the time zone of the place where the process is being run. Additionally, this concept can also be applied to things existing in different abstraction layers. For instance, an activity performed by an actor on the context of a business process (conceptual/business layer abstraction) might be dependent on a particular software system (system/logical layer abstraction).

A diagram showing how dependencies may cross through different conceptual layers of an enterprise is shown in Figure 1. Each layer can be thought of as providing a different perspective on an enterprise. The layers each provide a conceptual grouping of different elements and help to divide the perspectives of an enterprise into more manageable units. Relations naturally exist between elements in the same conceptual layer. For instance in the diagram in Figure 2, L1 could signify the top-level business process that could be represented using BPMN. Lower level layers such as L2 through to L4 could represent; service, software and infrastructure layers. The number of layers is not limited to four as shown in Figure 2 but depends on how many conceptual layers are decided upon. This again is a design decision. Many enterprises will separate the concerns of an enterprise into four or five layers based on the U.S. National Institute of Standards and Technology (NIST) representation. The NIST representation was adopted by the U.S. Federal Government in the Chief Information Officers Council's Federated Enterprise Architecture (CIO99) and is represented in Figure 1. As discussed in the approach in Section 3.1.2 an alternative framework for dividing the concerns, the Zachman framework is used.

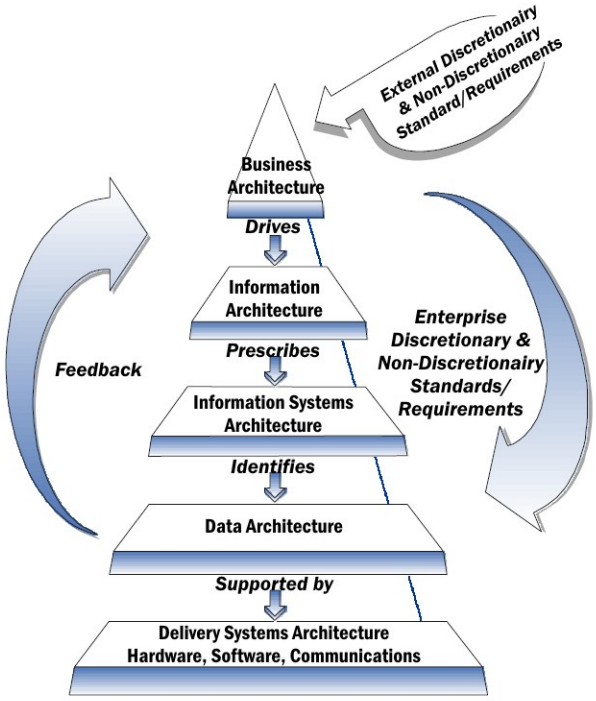

Figure 1: NIST89[1]/CIO99[2] layers of an enterprise architecture

Elements in the BPMN layer could then have a representation in the Formalism. These BPMN elements would then be connected through terms that are more appropriate for that particular domain. BPMN 2.0 for instance has several types of connections; sequence flow, default flow, conditional flow, message flow, conversation list, forked conversation links and associations.[3]

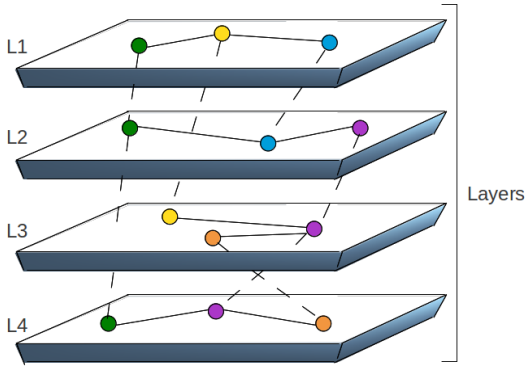

Figure 2: Dependencies throughout the scope of an enterprise

[1]http://www.itl.nist.gov/lab/specpubs/NIST%20SP%20500-167.pdf (Figure 7-1)

[2]http://www.cio.gov/documents/fedarch1.pdf

[3]http://www.omg.org/bpmn/Samples/Elements/Connections.htm

2.2 Problem Statement

Digital preservation aims towards the optimisation of the information life cycle management, from the creation to the dissemination and usage of digital objects, with the objective of maintaining the knowledge contained in the objects accessible for future users, beyond the limits of media failure or technological obsolescence.

The complexity of digital preservation increases with the fact that different organisational scenarios contain different types of objects, each with its own preservation requirements. This can be easily verified, for instance, when comparing the preservation of digital objects with static content, such as text files, with dynamic objects, such as executable process workflows that are used by workflow engines. Moreover, in these complex cases, the semantics and behaviour of objects can be largely dependent on the context where those objects are created and used, thus requiring a particular environment in order to be understandable or rendered. Indeed, the preservation of such objects also requires the preservation of multiple inter-dependent objects, without which it might be impossible to interpret.

For digital preservation to succeed, the various dependent objects that are required need to be identified. If this could seem a trivial task for objects that are explicitly dependent on each other, it becomes a challenge for objects whose dependencies are not easily observed (for instance, those representing conceptual entities belonging to different abstraction layers, as exemplified above). If not modelled explicitly, it becomes very easy to lose track of objects that are indirectly or not explicitly dependent on each other.

Additionally, in such scenarios involving complex dynamic objects that are sometimes subjected to changes also prompted by changes in environmental conditions (for instance, executable business process specifications), the monitoring of the impact of changes on the network of dependencies might be important to ensure the authoritative re-enactment of objects.

2.3 Goals

The overall goal of this deliverable is to provide a formalism that is sufficient to express the dependencies required for assuring the digital preservation of processes. For this purpose, existing, well established, or even standard Domain Specific Languages will be taken into account, so that the formalism is itself aligned with the best practices in dependency modelling existing in different domains (which will be surveyed in Section 3).

This formalism has been designed to model the parameters in an Enterprise relevant to answering the question “What elements need to be captured in order to digitally preserve a business process”. For that purpose, the formalism relates components on

different levels of the Enterprise (that usually are defined through their own DSLs, for instance BPMN for business process representations). These relations (dependencies) will then be formally captured and allow for reasoning to be performed. This will allow for the assessment of the extent to which an enterprise system and its processes can be preserved.

Reasoning is a form of solving problems using information that is either explicitly encoded into the problem or that can be inferred through a set of logical rules that apply to all elements in the system being reasoned upon. The usage of reasoning on the Formalism as defined will then allow, in concrete cases, for dependencies to be analysed between components that don't naturally have explicit relations or relations between components that belong to different abstraction layers. In this way information applicable to different abstraction layers (i.e., Business, System, and Technology) can be encoded in a way that is in line with domain specific languages but that also captures inter-layer dependencies. This will allow for the assessment of the impact of changes on a component at a determined abstraction level and how the changes will affect other components at different abstraction levels, triggering a digital preservation action as required.

For example, through reasoning the impact of changes to legal policies on the information systems of a specific type of organisation could be detected through the dependency network and based on the suggestions of the reasoner, the preservation of the systems and processes could be triggered. This helps contribute to preservation planning and risk assessment of the overall and constituent components of the business process. Besides being used for triggering preservation, reasoning could also be used for assessing if a process can be preserved, for instance when there is a component that cannot be preserved that is part of the dependency graph of a component that forms part of a business process then this model of the process would be marked as non-preservable. Non-preservability may be marked by either an expert or by the tools in cases such as;

- Cost of preservation of a component is prohibitive (in terms of man-power, cost and money). All restrictions of preservability derive from costs in some form.
- A business process expert might decide when examining the scenario that parts of the scenario are not essential to the overall business process. In TIMBUS this is combined with risk information that suggests the potential cost versus benefit of preserving specific parts of a business process and is performed in the IERM section.
- The technical feasibility of performing preservation may be restrictive in that capturing the original components of the business process may take a lot of human and computing effort.

- Legal issues may restrict preservation of parts of a business process due to incomplete licenses, contracts and agreements, amongst other issues.
- Business enterprise boundaries can also affect the preservability in that an enterprise may use resources and services from other enterprises that are outside of the control of the enteprise environment being preserved. Without preserving the connected services environment also, there is a limitation to the guarantee that the complete process is preserved. These may be alleviated by Escrow agreements that are being investigated in the scope of TIMBUS.

In this case, an alternative solution could be proposed if one is available and if all the solutions are assessed then process will be marked as non-preservable[4].

2.4 Approach

Dependency and component analysis and capturing can be done at various different levels. From the software and computing perspective this analysis can be done from low level instruction calls all the way through to top level networks of connected computers. From a business perspective, this analysis could be done from the level of a business actor performing a low-level activity up to the level or cross-organisation business processes.

The approach taken for this deliverable was to focus on the identification of software dependencies that are required for preserving the business process. Business and organisational dependencies will be approached in D4.3: *Dependency Models Iter. 2*, due in month 24 of the project. Despite that, this deliverable presents an extensive section (Section 3) on related work which also contains relevant references for business/organisational dependencies.

As already referred, various perspectives on the relevant context entities and relationships can be adopted, which each might have different components that must be preserved. To better manage the different concerns of an enterprise, some initial ideas for splitting the components were suggested and worked on. While working on the categorisation of components it was determined that this was not a trivial problem. To better categorise the concerns a well-established divide-and-conquer framework, the *Zachman framework*, was chosen for categorising the contextual parameters and entities. This framework is normally seen as the 'Enterprise Ontology' but has no exact categorisation definitions but rather provides a series of recommendations. It will be described in more detail later in Section 3.1.2.

After the identification of the relevant context entities, it was necessary to define, in a precise way, what is meant by a component or entity and also what is meant by a

[4]a property that can be assigned manually or automatically based on technical feasibility, risk and cost

dependency. A state of the art review of existing formalisms for the representation of entities and relationships was performed with the objective of merging the expert knowledge analysis performed by various systems into a single common formalism. This formalism defines how the entities relate to each other, both in terms of a specification of the entity within the formalism and also in terms of the relations that they have between themselves and other entities. Additionally, it can also be extended to support other perspectives of an enterprise architecture.

Following the definition of the dependencies, it was necessary to apply the developed formalism to a use case dealing with a Music Classification Process with the purpose of showing it has enough expressiveness to capture the properties of the technical infrastructure supporting such a process.

Reasoning is a complex issue and will be one of the main focal points in the subsequent Deliverable, D4.3. One of the main motivations for having a Formalism is that by having a reasoner connected to the formally captured concepts, automated reasoning tools are able to detect errors and infer information that may otherwise take considerable human intervention or may not be immediately intuitive. As this is a large subject that needs careful consideration it will be fully considered in D4.3.

2.5 Relationship with rest of TIMBUS project

The TIMBUS project gathers context information about the business processes that reside in a source execution environment in four primary ways. Figure 3 below illustrates these at a high level showing how service dependency analysis is related to the overall project. TIMBUS contexts can be logically grouped into tasks relating to iERM, service dependency analysis (both software and hardware), business process contexts and regulatory lifecycle management. These provide all the information necessary to fully describe a business process, its constituent components, how they fit together, why they need to be preserved and for how long and finally should also capture the knowledge required by a future community to step through the execution of an exhumed version of the original process.

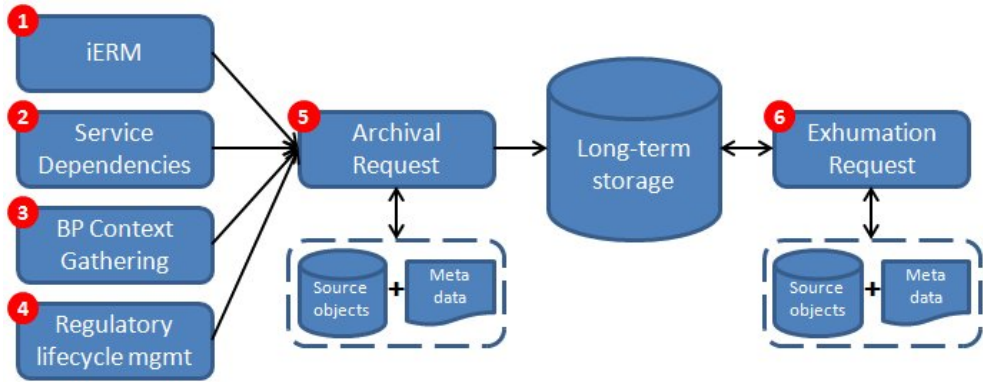

Figure 3: High-level view of TIMBUS approach

Service dependency analysis is an important part of this chain and contains the information needed to first identify all interdependent service components so that they can be captured for preservation and secondly this data is also needed to reconstruct the exhumed environment in the future. In general, TIMBUS Workpackage WP4 is concerned with designing and defining methodologies, ontology's and formalisms for the four context types shown in Figure 3. Workpackage WP4 takes its lead from the TIMBUS architecture which is designed in Workpackage WP5 along with the requirements of the execution tools in Workpackage WP6 and the use cases in Workpackages WP7, WP8 and WP9. More specifically, the output of this deliverable (D4.2) and D4.3 form the basis of the autonomous service dependency reasoning system for D6.2. This deliverable is also supported by T4.4 that contributes context modelling and reasoning aspects. These results help to contribute towards the overall exploitation plan that is described in Workpackage WP2 where the potential benefits to the industry and enterprises are being investigated. The investigations being carried out in WP2 are supported by those of the dissemination activities in Workpackage WP3 that attempt to highlight the results of the TIMBUS project to the research, scientific and industrial communities as well as to those who may not already be involved in digital preservation communities but could benefit from the results.

2.6 Document Structure

This deliverable is structured in 7 sections. Sections 1 and 2 refer to the Executive Summary and the Introduction respectively. Section 3 describes an extensive set of Related Work that approaches several areas of interest to the work presented in this deliverable:

i) Relevant references for the modelling of dependencies at the business, information, software, and hardware levels, which assumes particular

importance in the identification of the parameters and dependencies that are important to capture in this Formalism;

ii) Relevant references for configuration management standards and practices, which also deal with the identification and capturing of dependencies;

iii) Information capturing and data mining, which might be important sources for dependency identification and capturing;

iv) Relevant topology models that could be used as a basis for the Formalism.

Section 4 describes the resulting Formalism, more specifically the aspects involved in its construction, the language used as a basis for the model, the formal semantics for the reasoning, and how the versioning of the Formalism will be performed. Section 5 will then demonstrate the application of the formalism to a purpose-built use case that is described in detail in that section that is used for evaluation and iterative refinement of the Formalism. Section 6 contains a summary of the Deliverable stating what was achieved, the lessons learnt and the roadmap for the subsequent Deliverable D4.3 that will be used to continue the work carried out here.

The terms that will be used in this Deliverable are described in the Fundamental Concepts in Annex (Annex A.1 - Fundamental Concepts). In Annex (Annex A.2 - Dependencies in Service Operation and Lifecycle Processes), specific ISO standards relevant to Information and data life-cycle management are included that can be used for managing the evolution of data. Annex (Annex A.3 – Dependency relations as mapped by IBM Rational Software Architect) provides an example of a certain Industry tool that is (IBM RSA) referred to in the related work, Section 3, and has a different way of handling dependencies and is given as an example of a more limited alternative that is currently being used. Annex (Annex A.4 - Example Listing of CUDF for the TIMBUS Music Process in Taverna) gives a sample listing of the purpose-built use case described in Section 3.3.2 when captured using a set of tools using previous work of CUDF, described in Section 3.3.2. Annex (Annex A.5 – TIMBUS inverse relations mapping) is a listing of a set of the inverse relationships for those that are described in Section 4 and specifically Sections 4.5 and 4.6. Annex A.6 includes a listing of sample snippets of the representation of the Formalism, as described in Section 4, in OWL-RDF/XML format.

Section 7 then includes the references for materials cited in and concludes the Deliverable. If the citation is a URL it will not be included as a specific bibliographic reference but rather as a footnote in the corresponding section to allow the reader to investigate the particular subject in more detail.

Reading of the references and Appendicies is not mandatory but it may be useful to clarify certain concepts as required.

3 Related Work: Standards and Methodologies in Use

Modelling dependencies for TIMBUS can benefit from a range of established notational standards and methods applied for different purposes. When standards and methods prove useful in IT, supporting tools for the IT industry are developed. In this section, an overview of established and widely used standards and methods for defining and supporting business processes will be provided. The overview starts from an abstract view on organisations described by architectures in Section 3.1. Next, different standard approaches for the modelling business processes are described in Section 3.2. As business processes are usually supported by software services, the models for describing software dependencies are described in Section 3.3 beginning with the high level notations of business and software services to the detailed view including the description of low-level software dependencies applied on various operating systems. Afterwards, in Section 3.4 the modelling and capturing of hardware dependencies conclude at the lowest level of abstraction and complete the top-to-bottom dependency hierarchy. Typical software lifecycle processes and methods for extracting business related information are described in the Annex (Annex A.2 - Dependencies in Service Operation and Lifecycle Processes). These can be used to populate the models with concrete information and will be the starting point for linking the digital archive with the business to preserve. In Section 3.5, methods for extracting information depending on the level of abstraction are reviewed and in Section 3.6 the concept of ontologies is backed up by current related work. Finally, in Section 3.7, currently established preservation standards are linked with service dependency modelling concluding the overview of the current related work.

The overview provided in this section feeds into development and description of the complete formalism described in Section 4.

3.1 Modelling Organisations, Assets, and Processes

Modelling dependencies is an established tasks in IT and TIMBUS can make use of the models applied in software development. The modelling of dependencies at the business process and organisation level is a typical feature in enterprise architecture meta-models, although not referred to explicitly as dependency modelling. Enterprise architecture frameworks provide their respective meta-model with the aim of enforcing traceability between the strategic requirements of the organisation and the technological infrastructure supporting the business thereby promoting business/IT alignment. Two of the leading examples of such meta-models are the ones provided by The Open Group Architecture Framework (TOGAF), described in Section 3.1.1, and Zachman architecture framework, described in Section 3.1.2. These serve the

organisational embedding of processes and have their role in mapping artefacts or components to processes and organisations. Models defined and populated for the purpose of development can be exploited and re-used in TIMBUS to identify dependencies and explore the contexts of software services.

On the intra-process level, dependencies are usually depicted in business process modelling notations, such as the Business Process Modelling Notation (BPMN, see Section 3.2).

3.1.1 The Open Group Architecture Framework

Besides providing a method for the development of an enterprise architecture, TOGAF [TOGAF 2011] provides an architecture content meta-model. The meta-model defines the kinds of entities existing in an enterprise, at multiple levels, and the horizontal and vertical relationships existing between those entities, which could point to possible dependency relationships. The meta-model entities can then be instantiated in the development of concrete models of the organisation. The diagram in Figure 4 depicts the TOGAF meta-model, which enables modelling of intra-layer dependencies between the concept of *process* and other concepts belonging to the same conceptual layer.

Cross-layer dependencies between the concept of *process and other concepts* are also possible to be observed, namely through the concept of *business service*, which interfaces with “lower” layers. With the “upper”, strategic layer of *Architecture Principles, Vision, Requirements, and Roadmap* it is argued that the concepts belonging to that layer are related with all the other concepts of the layers below, with no explicit dependencies being enforced.

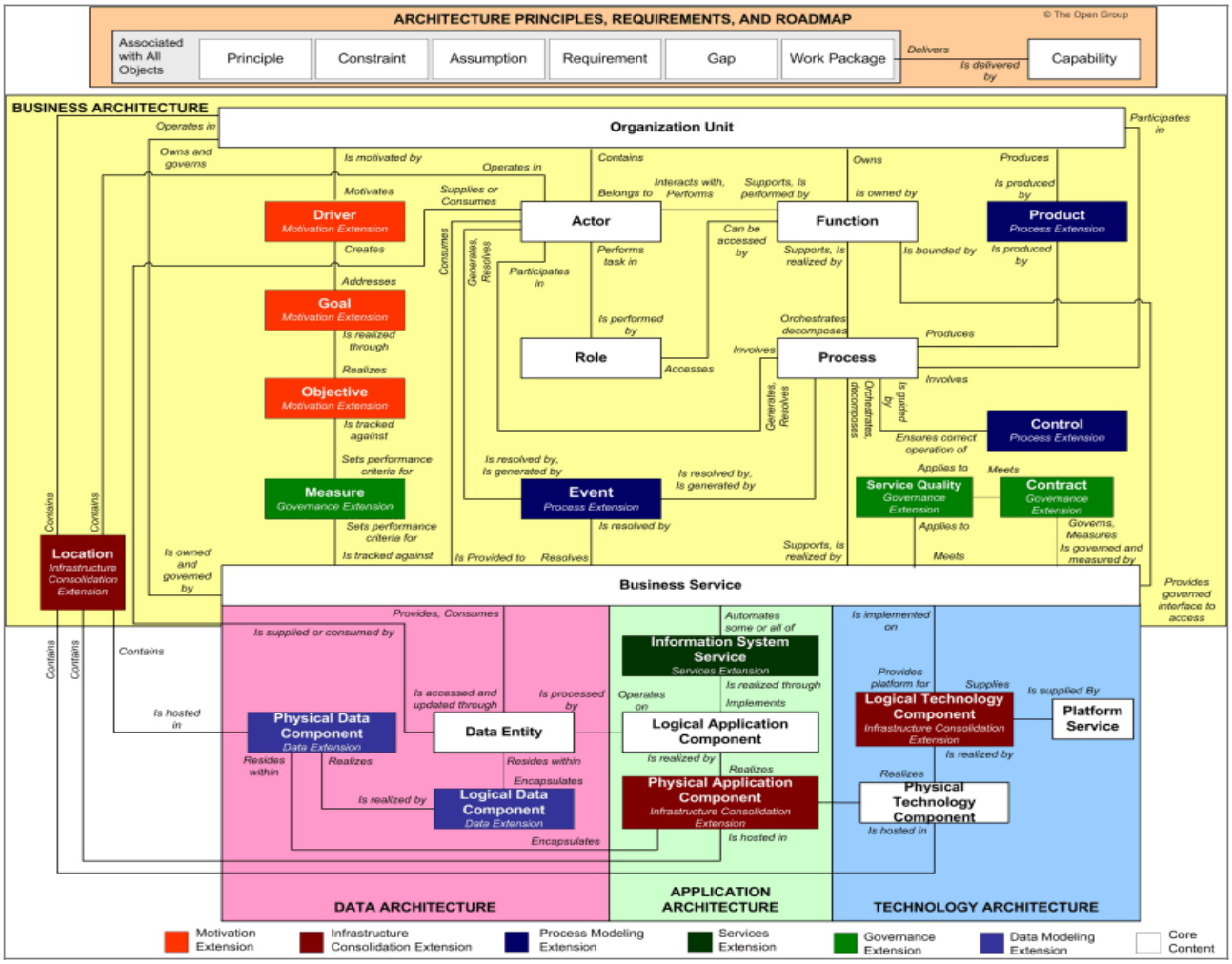

Figure 4: TOGAF Content Metamodel [TOGAF 2011]

3.1.2 Zachman

The Zachman framework [Zachman, J., 1987] was one of the first enterprise architecture frameworks created. It tries to take an holistic approach to the description of an organisation and the IT systems supporting the organisation's business purpose. The top-level entities addressed when using the Zachman framework are listed in Figure 5. The structure provided is used for defining the role of information systems in the enterprise, with the purpose of providing different views of the organisation with regards to different stakeholders[5].

[5]http://www.zachmaninternational.us/index.php/ea-articles/100-the-zachman-framework-evolution

Following the Zachman framework, a number of models, principles, services, and standards needed to address the concerns of one or more stakeholders have to be described. The “Scope” defines the business context, including the business purpose and strategy; the “Business Model” describes the organisation. Relevant dependency information are implied in the “System Model” describing how the systems will satisfy the organisation's information needs at high level and independent from concrete implementation. Similarly, the “Technology Model” describes the implementation of the systems and is expected to contain valuable dependency information at a system level whereas “Components” provide details each of the system's components and carry intra-system dependencies. Finally, “Instances” give a view of the functioning system in its operational environment and thereby disclose dependencies to partner systems, networks and hardware.

	DATA What	**FUNCTION** How	**NETWORK** Where	**PEOPLE** Who	**TIME** When	**MOTIVATION** Why
SCOPE (contextual)	List of things important in the business	List of business processes	List of business locations	List of important organizations	List of events	List of business goals and strategies
BUSINESS (conceptual)	Conceptual data/object model	Business process model	Business logistics system	Work flow model	Master schedule	Business plan
SYSTEM (logical)	Logical data model	System architecture model	Distributed systems architecture	Human interface architecture	Processing structure	Business rule model
TECHNOLOGY (physical)	Physical data/class model	Technology design model	Technology architecture	Presentation architecture	Control structure	Rule design
COMPONENTS (detailed)	Data definition	Program	Network architecture	Security architecture	Timing definition	Rule specification
INSTANCES (functioning enterprise)	Usable data	Working function	Usable network	Functioning organization	Implemented schedule	Working strategy

Figure 5: Zachman Framework [Zachman, J., 1987]

However, the Zachman Framework is a generic framework and makes suggestions on the types of models and contents that can be used for the various aspects. As there are no formal prescriptions, the analysis of dependencies strongly depends on the concrete implementation of the framework in the respective organisation. For concrete realisation of the framework, further methods and tools must be applied – a selection of which are described in the following sections.

3.2 Modelling Business Processes

Business processes exist in all types of enterprises and are described and documented in manifold ways. In order to reasonably describe and capture software services and their dependencies, organisations will make use of standards or best practises such as ArchiMate. They may also choose to describe the flow of the business process using defacto standards such as Business Process Modelling Notation (BPMN). Most enterprises find the procedures and methodology described in ArchiMate overly complex and will resort to a description using BPMN that only weakly describes relations in terms of which activities precede others and state who will perform those activities. When business processes are documented, higher level dependencies are documented and made explicit in different formats. In TIMBUS, explicitly documented business processes are a precondition for a complete and exhaustive dependency analysis – if the processes are only implicit, it will be difficult to assess the completeness of supporting software services and IT systems. In the following, two business process models are described that can serve as input for the software dependency analysis in TIMBUS.

3.2.1 ArchiMate

ArchiMate's [Archimate 2005] offering is distinct from that of TOGAF: while it does not provide a method for architecture development, it comprises a modelling language for enterprise architecture with an associated meta-model. The meta-model defines the entities that can be described within the blueprints and design of the architecture, which can exist at the level of business, application and technology, and in one of three perspectives: structure, behaviour and information. In ArchiMate, a business process is seen as a specialisation of the business behaviour element concept (which contains other specialisations such as business function, business interaction, and business event). From the observation of the excerpt of the meta-model for the business layer entities depicted in Figure 6, it is possible to observe the intra-layer dependencies.

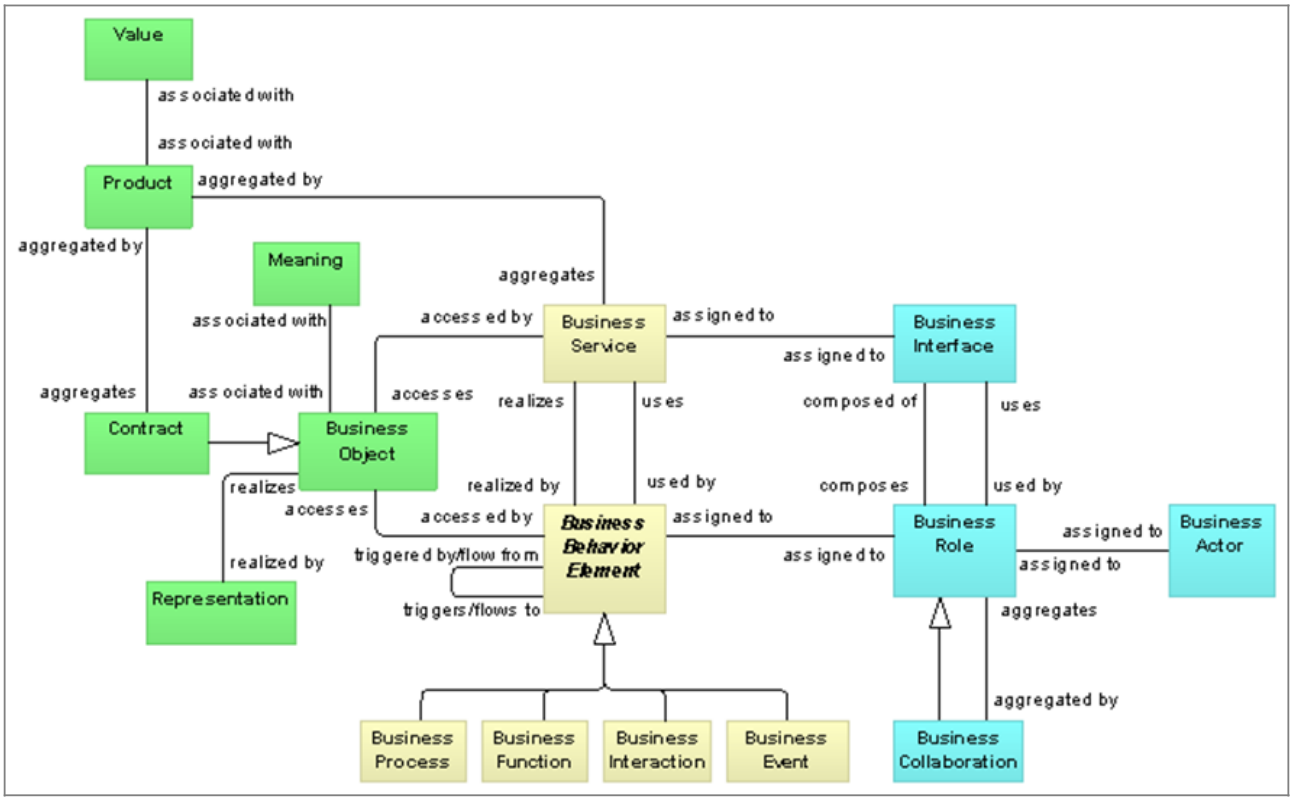

Figure 6: ArchiMate Business Layer Metamodel [Archimate 2005]

Cross-layer dependencies are also detailed in ArchiMate. In this case, the dependencies are between the business layer concepts and the layers below, since there are not any layers above the business layer. The cross-layer dependencies as captured in Archimate are depicted in Figure 7.

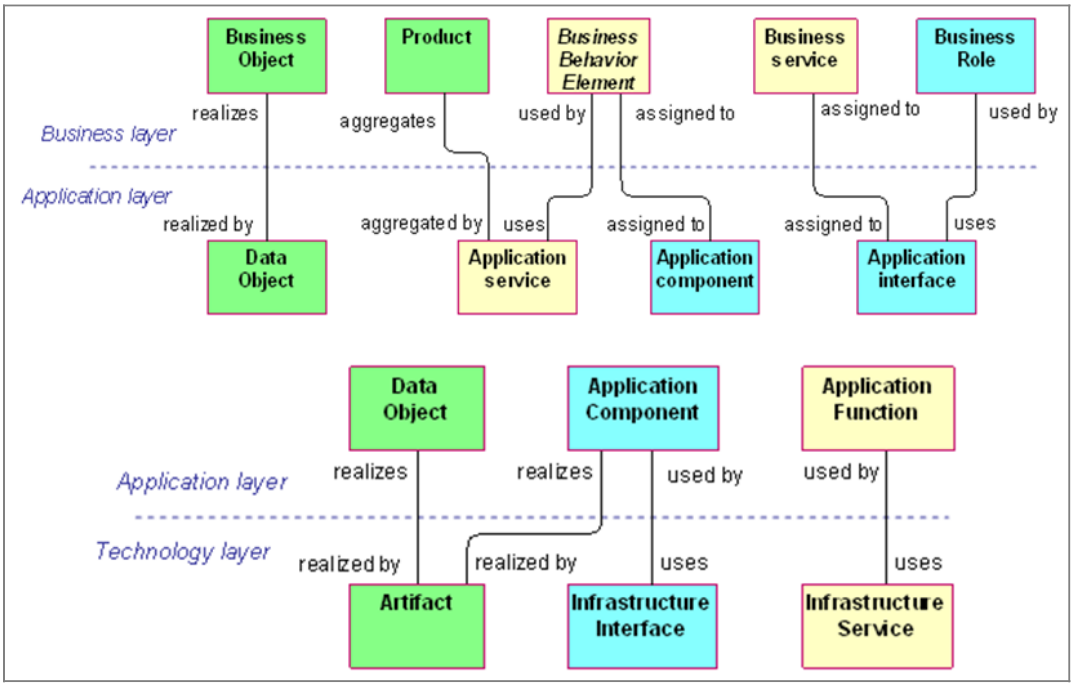

Figure 7: ArchiMate Cross-layer Dependencies [Archimate 2005]

3.2.2 BPMN

While ArchiMate models inter- and intra-layer dependencies, business processes also have intra-process dependencies. For instance, activities might depend on other activities, on certain events, or even on determined data. Business process modelling languages capture those internal dependencies that might matter to preserve.

The Business Process Model and Notation (BPMN) is a specification created by the Business Process Modelling Initiative (BPMI), first released to the public as version 1.0 in May 2004. The specification has since then been adopted by the Object Management Group, and it is currently on its 2.0 version [OMG-BPMN, 2011].

The motivation for using BPMN is to provide a notation from which all business users and developers can understand. BPMN creates a standardised bridge for the gap between business process design and process implementation. BPMN defines a Business Process Diagram (BPD) for creating graphical models of business process operations. BPD have three Flow Objects; Events, Activities and Gateways. Flow objects are connected together in a diagram to create a basic skeletal structure of a business process; Sequence Flow, Message Flow and Association.

When a higher level of precision is required for Business Process Management Systems (BPMS), additional elements can be added to the core elements and shown using graphical symbols. BPD also supports the concept of “pools” and “swim lanes”

to help separate functional activities into groups that are related. BPMN is designed to be flexible in allowing extensions to the notation but BPD currently only pre-defines three extension artefacts. There are two basic types of models that can be generated by a BPD: collaborative (Business-2-Business) and internal (Private) business processes.

As BPMN also helps to manage the difficulty of translating from the Business-oriented process modelling notation to IT-orientated execution languages, process descriptions in BPMN provide valuable input for process and service dependency analysis. Graphical objects of BPMN and a large set of attributes have been mapped to BPEL4WS, a process execution language.

3.3 Modelling Software Services Dependencies

Business processes are typically underpinned with IT services; IT services are composed from software systems. To this end, it is necessary to assess the dependencies between software components as to fully supplement the dependencies on business process and IT service level.

The purpose for using software dependencies in TIMBUS will be for us to be able to determine what software is required for preservation of a process. By computing the transitive closure of dependencies we can describe the minimal set of packages that are required for a software application. One caveat is that software dependencies tend to be related to local machines and the software installed on a single system. For setting up more complicated interactions where computers interact with each other over a network, the interfacing between systems is usually managed by configuration scripts or manager applications that handle the relations between systems. Generally though software dependencies on Linux systems are limited in scope to package management systems and as such when it comes to services and interactions with networks these are handled by external service dependencies such as web-services.

Software can be viewed at many abstraction levels from low level binary to much more complex higher level systems such as Operating Systems. Software dependency relationships on standard Windows machines and Mac OS machines tends to work at a high level, with software components requiring the use of an Operating System framework such as .NET[6] for Windows. There are other software artefacts that try to re-use commonly implemented systems such as Dynamically Linked Libraries (DLLs[7]).

[6]http://www.microsoft.com/net

[7]http://msdn.microsoft.com/en-us/library/windows/desktop/ms681914%28v=vs.85%29.aspx

3.3.1 High Level Software Service Dependencies

This section discusses software dependencies existing between remote machines, describing relevant service description languages. Service Description Formats are used to describe relations between software components on systems that are separated by the Internet or other medium. For TIMBUS purposes, these widely used languages again provide a starting point for dependency analysis and therefore are described briefly in the following.

WSDL

Web Service Description Language (WSDL) is an XML based language that is a W3C recommendation for describing the functionality provided by a web-service. The latest version at the time of writing is 2.0 [WSDL, 2007] that came into force June 26, 2007. WSDL is used for describing the functionality that a web-service may offer and as such is used with SOAP and XML schema to provide web-services over the internet. Accordingly, WSDL reveals dependencies to be considered during preservation.

WS-BPEL

BPEL4WS was a popular approach to Business Process Management using Web Services, submitted in 2003 to the Organisation for the Advancement of Structured Information Standards (OASIS) by BEA Systems, IBM, Microsoft, SAP and Siebel for standardisation. It is currently named WS-BPEL and it is currently on its version 2.0 [OASIS, 2007]. It is usually referred to using the moniker BPEL.

WS-BPEL can be used both to abstractly model a process and to create an executable business process. Executable WS-BPEL is essentially an XML programming language. A “program” in WS-BPEL is called a process. A process consists of a set of nested activities, which mostly fall into two sets; structured (allow sequential and conditional execution) and basic activities (invocation of external service to expose an interface to the process itself).

WS-BPEL defines a model and a grammar for describing the behaviour of a business process based on interactions between the process and its partners, which occur through Web Service interfaces, and the structure of the relationship at the interface level is encapsulated in what is called a 'partnerLink'. The standard defines how multiple service interactions with these partners are coordinated to achieve a business goal, as well as the state and the logic necessary for this coordination [OASIS, 2007].

WS-BPEL relies strongly on WSDL. The WS-BPEL process model is layered on top of the service model defined by WSDL 1.1. At the core of the WS-BPEL process model is the notion of peer-to-peer interaction between services described in WSDL, with both the process and its partners being exposed as WSDL services. A business process defines the coordination of the interactions between a process instance and its partners. In

this sense, a WS-BPEL process definition provides and/or uses one or more WSDL services, and provides the description of the behaviour and interactions of a process instance relative to its partners and resources through Web Service interfaces [OASIS, 2007]. In other words, WS-BPEL is used to describe the message exchanges followed by the business process of a specific role in the interaction.

WS-BPEL as discussed above reveals dependency information between partners and resources through web interfaces. These dependencies are also relevant for TIMBUS purposes.

UML

Unified Modelling Language (UML) is a standardised, general-purpose modelling language. It was created and is managed by the Object Management Group (OMG). It is now a de facto industry-standard that is used for modelling software systems. UML modelling uses conceptual components such as actors, activities, processes, etc. Also, it is extensible through the concepts of stereotype and UML profiles.

UML is currently at version 2.4.1 and is comprised of 14 types of diagrams, divided into two major categories: structure diagrams, with 7 different types, and behaviour diagrams, accounting for 7 types of diagrams of which 4 diagrams belong to the subgroup entitled interaction diagrams [UML, 2007]. Structure diagrams allow the modelling of the description of the entities, relationships and concepts of a system. Behaviour diagrams allow the modelling of the actions that the system can perform and the valid changes in the domain.

All this variety of diagrams allows the modelling of diverse viewpoints on a system, each depicting different kinds of entities and dependency relationships. In particular, *activity diagrams* are used to model the work-flows of the system; *use case diagrams* are helpful to identify the system's main functions and its relationship with the actors that interact with the system. These types of diagrams contain a lot of information about dependencies as described in this deliverable as they describe the interaction between objects.

SoaML

The Service oriented architecture Modelling Language (SoaML) is an UML profile and meta-model for the specification of service oriented architectures adopted by OMG. The first formal version is still under development, but a beta version dated December 2009 is available [SoaML, 2009]. Service oriented architectures are widely used in industry to structure enterprise software systems with the aim to increase re-use of individual software services as a level of abstraction. Dependencies are made explicit at a high level and can be leveraged as input for TIMBUS' dependency models.

With SoaML, a network of consumers and producers can be described in which each peer consumes or provides services to fulfil a purpose. According to the specification, SoaML supports the following capabilities: (i) Service Identification, including requirements and dependencies between identified services; (ii) Service specification, including the provided functional capabilities, the capabilities the consumers should provide, the protocols and rules for using the services, and the service information that is exchanged between consumers and providers; (iii) Service consumers and provides definition, including the services they consume and provide, how they connect, and how the functional capabilities provided by the services are used by consumers and implemented by providers in a way that respects the service protocols and requirements; (iv) Definition of policies for using and providing services; and (v) Service and service usage requirements definition and linkage to other OMG specifications.

BSDL

The Business Service Description Language (BSDL) has the purpose of describing business services from a pure business perspective, addressing specifically their decomposition and non-functional properties [Le, L. S., et al., 2010]. It aims to close the gap existing between more strategy and goal description languages and operational service description languages.

The modelling concepts provided by BSDL are categorised in five groups: Basic, which includes the basic concepts of the language; Functional, which includes all the concepts related to functional aspects of services; Non-functional, which includes all the concepts related to non-functional aspects of business services; Lexical, which includes all the concepts related to descriptions of business services and related aspects; and Decomposition, which includes concepts related to service decomposition.

The basic concepts of BSDL relevant for TIMBUS are: Business service, which represents a high-level service provided by a business entity; Provider, which represents a business entity that provides a Business Service; and Requester, which represents a business entity that requests a Business Service. The relevant non-functional concepts are: Obligation, which depicts mandatory responsibilities of requester or providers; Environment, which is an obligation related to environmental-friendly concerns. Obligation can be regarded as a specific type of dependencies and are as such relevant for modelling dependencies in TIMBUS. For matters of brevity, lexical and decomposition concepts will not be described.

3.3.2 Low Level Software Service Dependencies

This section discusses software dependencies inside a single machine and the current related work in relation to dependencies of software systems. The relevance for dependency models in TIMBUS lies in the massive number and manifold variations of software packages depending in different ways on each other. Even in virtualised environments for the execution of applications, the inherent software package structure has a key role when preserving business or software services: In order to execute a application the full set of required software packages needs to be in place – even the smallest missing piece of digital artefacts required for execution has the potential to render the entire application setup useless.

On a Windows or Mac system, there are still software dependencies but there is no packaging system for sharing re-usable components. Executables in these environments will link to libraries either statically or dynamically and are more dependent on system level frameworks or components. On Windows there is an installer called 'Windows Installer'[8] and on Mac it is 'Installer'[9].

Microsoft Windows Dependencies

On Windows systems the dependency system is very limited and as such software applications written for Windows tend to check for an Operating System version and state whether they can work based on version comparisons. This can lead to cases where software that should be able to run cannot as there is a version number that was not around at the time of development of the software. Software components therefore do limited dependency checking in an ad-hoc manner.

For Windows, Dependency Walker[10] checks which modules are called by other software that is currently being executed. There exist a few other tools for extracting software dependencies from Windows Systems but these are limited and not as expressive as their Linux counterparts[11] and for .NET dependencies[12][13]).

Mac OSX Dependencies

Mac OSX is more interesting as it is based on a BSD-Unix type system. At the Mac OSX level there is a similar limited dependency system that is used in an ad-hoc fashion. At

[8]http://msdn.microsoft.com/en-us/library/cc185688%28VS.85%29.aspx Windows Installer

[9]http://developer.apple.com/library/mac/#documentation/Darwin/Reference/ManPages/man8/installer.8.html MacOS Installer

[10]http://www.dependencywalker.com

[11]http://www.ucware.com/apev/how-to-view-dll-dependencies.htm

[12]http://www.reflector.net, http://www.ndepend.com

[13]http://tcdev.free.fr

the base of Mac OSX is a UNIX style system that has a similar dependency system to that discussed in the Linux/CUDF section.

Mac OSX runs on Darwin which is a UNIX style system that is a derivative of 4.4BSD-Lite2 and FreeBSD. BSD runs with binary 'ports' as packages which are then installed with pkg_add -r package[14].

During normal operation the dependency system of Mac OSX is not visible to the end user. As such during much of the operation of Max OSX, the dependency system is not visible to the end user and software components are handled in a manner similar to that of Windows. If the underlying system is modified then the system does have a dependency checking system and this acts in a similar way to that of traditional Unix/Linux systems.

A mapping into a high level view of software dependencies denoted in mathematical notation has been performed[15]. More information about the description of dependencies and component based software systems are also available from previous EC Project, MANCOOSI.[16][17][18] Different solvers work in different ways for reaching a solution and there have been many International Competitions to rank and compare their performance[19] but at the base they all use the same formalism and problem sets as their inputs.

Linux dependencies and CUDF

Linux systems were developed not by one company or a closed eco-system but rather in an open manner. As such the development could have grown in a manner similar to that of Windows and Mac OS where the system components are fixed and all the applications bundle all the necessary material to make the underlying system work with that software application. Instead the development made use of re-usable software components at the package level. Packages are small software components that typically encapsulate the binaries, resources and meta-information to perform a single function. These functions are well defined and then software developers build other software applications to match the interfaces as specified. A well-structured and defined dependency relationship model must therefore be adhered to and this is enforced by the use of a package management system. Developers can of course avoid using this and build their own software components independently that do not

[14]http://www.freebsd.org/ports/index.html

[15]http://people.debian.org/~dburrows/model.pdf

[16]http://mancoosi.org/reports/d3.1.pdf Description of component based systems

[17]http://mancoosi.org/reports/tr1.pdf DUDF Description Format

[18]http://mancoosi.org/reports/tr3.pdf CUDF Description Format

[19]http://mancoosi.org/misc/

adhere to dependency relations, but then they must build software from scratch. For continually evolving systems such as Linux this does not make sense.

On Linux and Unix-based systems the development model followed was that of component based engineering as discussed in [Szyperski, C. 2002]. This model of development led to packaging systems being developed [Mancinelli F, 2006]. These packaging systems maintain software as components known as packages. Packages are generally the smallest subset of data/resource, configuration and executable files such that they provide a feature. Packages contain the relevant application and also meta-data with information about what package dependencies they have. Package installers such as RPM (RPM Package Manager) and dpkg (Debian package installer) can read the packaging meta-data to infer if a package can be installed or removed. More complex relations between multiple packages are determined through Package Management Systems that read the meta-data in multiple packages that are scheduled for installation, removal or upgrades and can determine if the packages can be installed (satisfiability problem). If a simple solution is not available, depending on the algorithm and heuristics used in the solver, an alternative solution may be proposed. This may involve removing packages (that are in conflict) from the original system or require that other packages are installed in order to reach the desired solution (apt-pbo is an example of a solver[20]).

The most recent form of describing generic Linux dependencies, CUDF is now described.

CUDF (Common Upgradeability Description Format) [Treinen, R., et al., 2008] and DUDF (Distribution Upgradeability Description Format) are formats for describing upgrade scenarios in package-based Free and Open Source Software (FOSS) distributions. CUDF is one of the most recent attempts to capture software dependencies completely on Linux systems. It is the result of a previous EC Project-MANCOOSI[21]. CUDF was designed to capture and express upgrade problems in a format that is independent of the type of GNU/Linux Operating System and allows for a class of software tools, known as solvers, to work on identifying possible solutions for upgrading a set of packages requested by the user. It uses the package management systems at the base of the majority of Linux systems; dpkg for Debian type and rpm for Redhat type systems. These package management systems are then queried for certain information about the state of the packages as currently found on the system, the packages available to the installer (the package universe, normally on a remote webpage as a repository) and the request that the user or machine has requested (upgrade request/manifest). The DUDF format compliant request is generated on a per distribution basis. This is dependent on the type of installer that is being used on the

[20]http://aptpbo.caixamagica.pt

[21]http://www.mancoosi.org

distribution. The tools identify the corpus of packages that are available to the installer at the time the upgrade request is made. The full requests are then captured in a standardised manner and submitted to a centralised server based on the distribution. The distribution servers collate the information and convert the DUDF files into CUDF. The CUDF files are more abstract and describe relationships between packages at a higher level than DUDF. The CUDF representation is then submitted to a centralised repository where the upgrade problem sets can be collected. For TIMBUS, the upgrade request, when capturing the state of the system, is less important and the reasoning system will be used to infer whether certain dependencies are met.

3.4 Modelling and Capturing Hardware Dependencies

All IT supported business processes and software services require hardware at their lowest level for execution and this holds true for Virtualised Operating Systems. In order to capture artefacts to be preserved effectively, we therefore have to consider hardware and hardware dependencies accordingly. For TIMBUS purposes, it is useful to have the complete set of hardware dependencies identified prior to archiving and document the dependencies in the archive such that in the case of exhuming business processes the entire software-hardware set can be can be understood and recovered accordingly.

Hardware dependency analysis is a subset of the larger area of inventory and asset management. Any large enterprise organisation or IT department can expect to be using tools available today to aid with asset management such as SAP Enterprise Management[22], IBM's Enterprise Asset Management[23] and Xasset's Asset Management[24]. These are essentially inventory tools which will automatically scan all devices on the network to build up a map of the IT landscape. The primary motivators for these would include licence auditing, configuration and policy management, and asset tracking.

Existing tools work well for established and commonly used enterprise operating systems and application suites but are less effective for bespoke or in-house developed applications.

A further limitation is the lack of a robust capability to determine the hardware dependencies for specific operating systems or software prior to deploying it for the first time. In Section 3.4 of this deliverable, the idea of hardware compatibility lists (HCLs) for operating systems was mentioned but a typical OS deployment today

[22]http://www.sap.com/solutions/enterprise-asset-management/features-functions/index.epx

[23]http://www-01.ibm.com/software/tivoli/solutions/asset-management/

[24]http://www.xassets.com/asset-management-software.aspx

happens over the network and will only fail during the install process, not prior to it, in the case where the hardware is not on the HCL.

Hardware discovery, as part of inventory scanning, is carried out in several ways and the approaches can be different depending on factors such as the functionality provided by the hardware vendor, the choice of operating system and the choice of scanning tool (commercial, open-source, in-house developed). The following section will describe the practices used in industry today. However, it should also be noted that none of these allow for the identification of anything except the most basic of hardware dependencies such as requirements of available disk space.

Hardware vendors often provide supplementary manageability tools to aid in distinguishing their offerings in the market. These can be based on non-proprietary standards such as support of Simple Network Management Protocol (SNMP) or they can be hardware based, for example Intel® vPRO™,[25] and Intel® Remote Management Module[26] (RMM), both based on Intel Active Management Technology[27] (iAMT).

3.4.1 Capturing based on Non-proprietary Standards

Protocol standards such as SNMP[28] can be used to exchange device configurations. SNMP uses an extensible architecture which does not seek to define all the types of management metadata that can be exchanged by compliant devices. Instead data describing device configurations can be designed and exchanged for management purposes. It relies on a central management device and an agent which runs on each managed device. Network devices typically support ICMP router discovery capabilities as part of the definition of RFC 1256. Some inventory scanning tools may rely on the use of such standards to discover information about the hardware environment as part of IT operational support or auditing.

Other relevant standards in this area exist. For example, FIPA (Foundation for Intelligent Physical Agents)[29] is a standards body which was founded in 1996 but has been part of IEEE since 2005. FIPA has developed standards and ontology's which control communications between software agents running on diverse physical devices.

[25]http://www.intel.com/content/www/us/en/architecture-and-technology/vpro/vpro-technology-general.html

[26]http://www.intel.com/content/www/us/en/server-management/intel-remote-management-module.html

[27]http://www.intel.com/technology/platform-technology/intel-amt/

[28]http://tools.ietf.org/html/rfc3411

[29]http://www.fipa.org/

3.4.2 Capturing based on Hardware

The biggest advantage of hardware solutions for capturing device data is that they do not require the operating system to be online to provide configuration and management functions. These devices are primarily used to aid in remote support or Operating System deployment rather than for inventory purposes. Not only can they report detailed hardware specifications, but they can also provide remote sensor data and they can send alerts about failed system components.

3.4.3 Capturing based on Operating System

Operating Systems often contain some built-in mechanisms for gathering system configuration information via in-house developed scripts. In the case of Microsoft Operating Systems, this is primarily supported by WMI (Windows Management Instrumentation) and Windows Script Host (WSH). UNIX operating systems have always inherently supported these features through Perl, shell scripts and utilities.

3.4.4 Capturing based on Scanning Tool

To overcome the work-intensive maintenance of scripts, there is a wide range of commercial and open-source scanning tools at hand. These applications focus on gathering system configuration data for Windows, UNIX and other types of devices. They are used for a multitude of administrative tasks and could be reused for TIMBUS purposes to some extent. The tools are typically capable of gathering hardware information in whatever level of granularity is available to the user-mode processes.

3.5 Extraction of Information

In order to effectively preserve information, automatic or semi-automatic data capturing mechanisms should be in place, so that any associated context information required for its correct rendering and processing is gathered so that it can be preserved together with it.

The implementation of the context and data mining tools is left for deliverables D6.2 and D6.5. This section discusses general challenges in capturing data and information, describes how to extract information from business processes and, at the most detailed level, how to retrieve information from IT systems.

There is no general solution to the extraction and capturing task. The concrete methods and techniques depend on the respective contexts and differ in business domains and IT areas.

3.5.1 Data and Information

The term *information* should be stressed a lot because information are the essence describing the processes. Unfortunately, the terms *data* and *information* are often used synonymously, but the concepts as used in TIMBUS is explained in Annex (Annex A.2 - Dependencies in Service Operation and Lifecycle Processes). The distinction between data and information is substantial and can be made by semantics. Semantics transforms data into information. The well-established information model in information science makes the distinction like it is illustrated in Figure 8 (cf. Krcmar, H., 2005). The bitstream is turned into data when a specific syntax is postulated and the bitstream can be interpreted by an application like a relational database or a word processor. Adding intellectual background (ie., context and semantics) then turns data into information. Having this in mind, it becomes clear that the long term archiving approaches also require to preserve syntax and context to be capable of re-interpreting data with the correct semantics in a specific context.

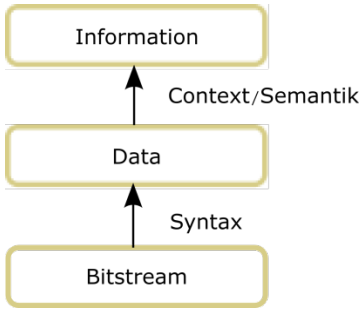

Figure 8: The research information model

3.5.2 Data persistence variants and challenges

Flat files

Different methods for persisting data/bitstreams have been established. The report in [Knijff et al., 2011] provides an evaluation of several file characterisation tools, some of which use the services provided by PRONOM. A lightweight approach for storing data is the usage of so called *flat files* in the file-systems. An example of this are the widely established local storing mechanisms offered by the “office suites”. The concept is well-suited even for complex digital artefacts. Flat files are not limited to local file systems but can also make use of shared file systems or other storage media. It is an established paradigm for storing data. But the concept poses some challenges especially with a view on digital preservation. The initial challenge is the different interpretation of bit streams in the different operating systems and language packs. Another issue is the complexity in identifying the application needed to interpret a

special file. Services like it is offered by the PRONOM initiative[30] deal with this challenge by offering information about numerous file formats.

Techniques like hyperlinks increase the complexity of files. Hyperlinks are offering a mechanism for making dependencies between files explicit. Furthermore, the "enhancement" of files with scripting languages like VisualBasic for Applications or JavaScript increases the complexity of files.

The capabilities of modern file systems like versioning mechanisms and search functionalities are upcoming challenges for the digital preservation, because they change the behaviour of software and define new hard-to-detect dependencies between different applications.

Databases

In contrast to the lightweight approach of a file system stands the more complex concept of databases. A database offers mechanisms for structuring data and manage it, for ensuring the integrity of data and optimising the performance of queries. One commonly used concept is currently the concept of relational databases. However, database are well accepted approach for managing big sets of relatively small data. All kinds of databases offer powerful mechanisms for providing structural information and within the explication of dependencies. The particular challenge in extracting data is that most of the systems are extended with vendor specific functionality which negatively influences the portability.

Cloud storage

A currently hyped topic in persisting is "Cloud based storage". A lot of research as well as a growing interest in all business sectors can be observed at this time. Cloud services can be categorised by their service layer and are then called *X*aaS (with *X* being the respective service layer, e.g., infrastructure, software etc) (c.f. [Rimal, B.P., et al., 2009]). The challenge for digital preservation of cloud services is that the service is opaque for the user. This means that only the service layer can be accessed and typically no further details are revealed. For example if a user wants to use a relational database in a Platform-as-a-Service (*P*aaS) environment, he has no access to the database configuration or to the physical database files. This problem area is similarly present for all cloud service levels and is not restricted to the area of data capturing but will influence the understanding of the preservation of the data context in general.

[30] http://www.nationalarchives.gov.uk/PRONOM

Big Data

Another upcoming challenge is the handling of big data. The easiest definition made by Edd Dumbill is: “Big data is data that exceeds the processing capacity of conventional database systems. The data is too big, moves too fast, or doesn't fit the strictures of your database architectures.“ [Dumbill, E., 2012]. With big data, all approaches like databases, files and cloud are brought together and seen in an holistic way. Illustrative examples for big data are the established social networks or business intelligence solutions which integrate different heterogeneous data sources. Big data brings uncertainty into computing in that it is neither necessary nor even possible to process all data in a deterministic way but enough data for making feasible decisions.

To summarise, all described kind of data and consequently information needs specific methods for capturing. Capturing information is more than simply copying well understood files into an archive. With the increasing complexity of the types of data storages the complexity of preserving information is also expected to grow.

3.5.3 Extraction of Business Process Information

The gathering of information regarding business processes is an highly relevant topic, e.g., with regards to performance or compliance concerns [Sadiq, S., et al., 2007] [Ghose, A., et al., 2007]. It is obvious that methods for extracting the current state of processes are necessary to preserve the processes. Dependencies from business process are crucial for the preservation of business process to achieve a complete picture of what is to be preserved.

The area of Business Process Management (BPM) has the aim of providing support to business process by the use of methods and techniques to design, execute, control and analyse business processes, provided that there is sufficient information to make them explicit [Van der Aalst, et al., 2003]. The BPM area includes the area of workflow management dealing with the design, configuration and enactment of processes, additionally including diagnosis dealing with the analysis and improvement of processes. One way of capturing business process data is through the use of formal modelling languages for the design of business processes for reasons of documentation or for execution. In practise, although design approaches are quite popular, they sometimes fail in capturing the reality [Van der Aalst, et al., 2007].

One of the ways of collecting information about business processes is through process mining. Process mining is a combination of approaches to capture process information, such as modelling and simulation, which might present a not so accurate or even distorted view of reality. Process mining has the aim of discovering process, control, data, organisational and social structures using event logs generated by a wide range of enterprise applications [Van der Aalst, W. M. P., et al., 2007].

Event logs contain time stamped information about events happening in a system. Each event can be related to an activity in the context of a process and has a performer. By collecting that information from several enterprise systems, causal and dynamic dependencies between events can be captured and lead to the automatic construction of business process models that truly reflect reality.

Typically, it is possible to assume three different perspectives concerning the information present in event logs. Firstly, the process perspective (also referred as the "How?"), which focuses on the control flow can be reconstructed; secondly, the organisational perspective (also referred as the "Who?"), which focuses on the performers of the activities involved in a process can be derived; and lastly the case perspective (also referred as the "What?"), which focuses on the properties of the cases, or process instances (e.g., the number of products ordered in an buy order) can be mined. However, process mining and its associated techniques also suffer from problems, such as noise and exceptions, which might render a process model incomplete [Van der Aalst, et al., 2003].

3.5.4 Runtime Information extracted from IT systems

More detailed information can for example be mined from Operating Systems. The possibilities for extracting information from the operating system strongly depend on the level of transparency of the system. In this case open source system have a strong advantage. Because the source code is available (or at least the interfaces are well documented and standardised) it is relatively easy to determine most of the information through scripts and system-calls in operating systems like Linux, BSD or the other UNIX-compatible systems. This includes the dependencies of the different components.

The following illustrates some simple example of how information can be captured. The hardware if recognised by the system and enumerated can be easily captured as well using standard methods such as "lshw". The file system mounts as found on the system normally can be found in "/etc/fstab". Information about other hardware can be found under "/dev/" and "/proc/" on most Linux machines. Other methods can be used for determining the kernel details such as version "uname -r". Most of these methods are well established and have been used by system administrators for managing systems. These calls and scripts can be bundled into management systems and can also be run by data-miners.

This information can be used for generating the context of individual machines by investigating the hardware and configuration on individual systems. It is also possible to use package management systems to determine what software is installed on a system and to query that database to see which packages are installed, see also Section 3.3.2.

Data mining can also be done remotely through passive and active methods. Most passive methods rely on sniffing Internet traffic that passes through a network medium and analysing the packets to see what type of data is being carried in them. This has led rise to many applications that can determine various amounts of information by passively viewing data on a network[31].

Active methods can also be used to attempt and trigger certain behaviours on remote systems and depending on the behaviour it can sometimes be determined what sort of system is responding through its characteristic responses. There are some open-source data mining tools for software and applications such as the Java machine learning library of WEKA[32].

On closed source systems like Mac OSX, BeOS or Windows it is more difficult to extract all the useful data as different flavours of the operating systems come with different methods for storing and managing the information about a system and not all of them are well documented. In Windows a lot of the configuration and settings data for the system and its applications are stored in a key-value pair repository also known as the Registry. Different systems may have different ways of storing data and this soon becomes a difficult problem to manage and as such there were attempts made to use Policy Managers that used established methods for retrieval of data. This situation is somewhat improved by formal, standard methods and protocols for capturing information. Most of the time though, Enterprise solutions need to maintain mechanisms for interacting with old OS' and as such will go through a series of steps to attempt to determine what type of system is on the Network. Sometimes though support is removed and as such it can sometimes be difficult to retrieve information about legacy systems through modern OS. An example of a change that affects users and administrators is the location of the User folder. Between successive iterations of Windows versions it moved between being at “C:/Documents and Settings/” to “C:/Users” etc. When storing data some of this information would be hard coded into old software and as such causes problems with new Operating Systems and hence is the reason why some systems such as Windows 7 have a legacy XP mode for allowing older software to run that has the context of the old environment. This is used to help users who have migrated to new OS' but cannot use older applications due to them not being migrated to the new OS.

To summarise, there is a cost associated with data and information gathering. For certain components it may be more effective for collating data as it may be more accessible and require less effort for mining. Extracting the context of the whole system results in total preservation as all the information is gathered. For digital preservation of a business process it may be acceptable for a less complete set of

[31]http://nmap.org/book/osdetect.html

[32]http://www.cs.waikato.ac.nz/ml/weka/

information to be gathered. If a lower level of accuracy is acceptable, on-demand virtualisation and preservation of the entire system may be a feasible alternative, because it captures the current state of a machine, all dependencies are included.

3.6 Information modelling

Information modelling is associated with topologies and ontologies. In general, *topology* is a term used when describing the structure or form of an object, e.g. mathematical topology, network topology, software topology. With regards to dependency analysis, software topology models, which are used to describe elements and interconnections of enterprise software systems, are particularly of interest. For simplification reasons, software topology models are in this deliverable simply referred to as topology models.

A similar addressed challenge is the preservation of the object's meaning. Especially the preservation of today's tacit knowledge has to be addressed. An initial step is the modelling of the knowledge. This can be made by using ontologies which are strongly related with topologies. An ontology formally represents knowledge as a set of concepts within a domain (a standard approach using taxonomies for this purpose), and the relationships between those taxonomies.

There are several topology models and ontologies that cover different aspects of enterprises. Each has its advantages and disadvantages as well as its specific areas of focus (domain specialisms). In the next sub-section we discuss a well established standard used for modelling semantics RDF/OWL. An example of a tool that can be used for modelling the relations between software components in an enterprise environment is IBM Rational Software Architect whose system is described in more detail along with the description of the relations that it models, in Annex (Annex A.3 – Dependency relations as mapped by IBM Rational Software Architect). Also another related work is that of the OAIS informational model, which as it covers specific digital preservation issues is in Section 3.7.1.

3.6.1 Enterprise Ontology

The Enterprise Ontology, is a collection of terms and definitions relevant to business enterprises. It was developed as part of the Enterprise Project, a collaborative effort to provide a method and a computer toolset for enterprise modelling.
The Enterprise Ontology is composed by a set of entities and relationships between entities. Entities can have roles in relationships. An attribute is a special kind of relationship and a state of affairs a situation which is characterised by a combination of entities in any number of relationships with one another [Uschold M., et al., 1996]. The main relationships of the enterprise ontology are depicted in Table 1.

Table 1: The main relationships described in the Enterprise Ontology

Type	Description
Entity	A fundamental thing in the modelled domain. Examples: (1) a human being; (2) a plan. An entity can participate in relationships with other entities. There is no distinction between a type of entity, and a particular entity of a given type. The word entity is used with explicit reference to a certain thing, but most of the references to entity in this ontology implicitly define a category or type of entity.
Relationship	Is the way two or more entities can be associated with each other. Examples: (1) The have-capability is a relationship between a person and an activity denoting that the person is able to perform the activity; (2) a sale is a relationship constituting an agreement between two legal entities to exchange a product for a sale price. A relationship is itself an entity that can participate in further relationships The words relationship has many meanings in natural language. The following meanings are important but logically distinct concepts that relationship commonly refers to: 1. The kind of relationship (closest to above definition); 2. A name given to the kind of relationship (e.g. marriage); 3. A particular relationship between particular entities. Examples: (1) Bill and Hillary Clinton are in a marriage relationship; (2) Einstein was in a have-capability relationship with the activity of thinking. Further distinctions can be made reflecting the use of the mathematical concept of tuple.
Role	Is the way in which an entity participates in a relationship. Examples: (1) Vendor is a role played by an entity in a Sale relationship. A participating entity is said to be playing the role. Strictly speaking, the correct way to refer to an entity playing a particular role is to use a phrase like ‘the Entity playing the Vendor role’. This is awkward,

Type	Description
	and instead, we will often use the shorter phrase ‘the Vendor’.
Attribute	Is a relationship between two entities (referred to as the ‘attributed’ and ‘value’ entities) with the following property: within the scope of interest of the model, for any particular attributed entity the relationship may exist with only one value entity. Example: Date of Birth is an attribute associating only one Date with a given Person. From a mathematical perspective, an attribute is a function.
State of affairs	Is a situation; the following is necessarily true of a state of affairs: (1) it consists of a set of relationships between particular entities (E.g. ‘Joe Bloggs can lay bricks’ (i.e. is in the Have-Capability relationship with the Activity: bricklaying.’)); (2) it can be said to hold, or be true (and conversely to not hold or to be false).
Achieve	Is the realisation of a state of affairs; i.e. being made true. When the state of affairs is a purpose, one would frequently say it is being ‘accomplished’.

3.6.2 TOVE Project

The TOVE project, acronym of *Toronto Virtual Enterprise* project is a project to develop an ontological framework for Enterprise Integration (EI) based on and suited for enterprise modelling. In the beginning of the 1990s it was initiated by Mark E. Fox and others at the University of Toronto.

The basic entities in the TOVE ontology are represented as objects with specific properties and relations, these relations are depicted in Table 2.

Objects are structured into taxonomies and the definitions of objects, attributes and relations are specified in first-order logic. The ontology is defined in the following way; TOVE first identifies the objects in the domain of discourse that will be represented by constants and variables in TOVE's syntax [Fox M., et al., 1997]. Subsequently the properties of these objects are identified as well as the relations that exist over these objects and these are represented by predicates in TOVE.

Table 2: The main relationships described in TOVE

Type	Description
Goal	TOVE models organisation goals that can be decomposed into an AND/OR subgoal trees, and can be achieved by executing activity clusters.
Organisation agent	An organisation-agent is an individual member, a human being, in the organisation. The concept of organisation-agent can be extended to include machine agent or software agent if needed. An organisation-agent is a member of some division, plays one or more roles in the organisation, can perform activities, and communicate with other organisation agents using communication-links.
Division	Each organisation-agent is member of or affiliated with some division (or sub-division) in the organisation. In the model each agent is a member of some division: Each organisation-agent is member of or affiliated with some division (or sub-division) in the organisation.
Team	An organisation-agent may also be a member of some teams set up to pursue specific tasks in the organisation. Compared to division which is usually a long-term setup within the organisation, team is temporary in nature and is usually set up when needed. Members of a team may be from different divisions and there may be many teams set up in the organisation. The relationship between an agent and a team is 'member of', which means an agent is a member of a team.

Type	Description
	Only two or more members can form a team. A team, as a whole, can play a role in the organisation. If everyone in a team plays a same role, we also say that the team plays the role.
Communication-link	Communication-links are established among organisational agents in various roles. Communication-links capture the notion of benevolent communication in which agents regard each other as peers volunteer information that they believe relevant to other agents. This exchange does not create obligations for any agent. The communication-link is a unidirectional link used to communicate information from one agent to another. It describes, for an agent in a given organisational role, the information it is interested in receiving and the information it can benevolently distribute to others For example, an agent in the “C++ programmer” role may distribute information about the state of the file server to other programmers, alerting them each time the server is down. The communication-link specifies: (1) Sending-Agent, the agent sending information along the link; (2) Receiving-Agent, the agent receiving information from the link; (3) Sending-Role, the organisation role played by the sending agent; (4) Receiving-Role, the organisation role played by the receiving agent; (5) Interests, the information interests of the receiving agent; (6) Volunteers, the information the sending agent can supply to other agent.
Authority	A special kind of authority is the control relationship between two organisational agents (OA). For OA1 to have authority over OA2 implies that OA1 is able to extract a commitment from OA2 to achieve a goal that is defined as part of OA2’s organisation-roles. In order to extract that commitment, OA1 has to be related directly or indirectly by a communication-with-authority link relation. The Communication-with-Authority link, used when communication is intended to create obligations, specifies the two agents, one in the authority position (called supervisor) and the other in the controlled position (called supervisee), among which communication takes place. Because we model communication as exchange of speech-acts, authority of an agent appears as the set of speech-acts this agent can use in order to create obligations for the other agent. For example, an agent may have authority to request another agent to perform action A1, but not to perform

Type	Description
	action A2. In this case, the second agent will have to commit to achieving A1 when requested by the first agent, but not A2.
Commitment	We introduce the concept of an OA's commitment to achieving a goal. The predicate 'committed to' signifies that agent is committed to the achievement of goal. Consequently, the totality of activities performed by the agent must include the achievement of the goal. Prioritisation of goals, etc. are not considered here. Any agent that fills an organisation role is committed to the goals associated with the role. An agent can only allocate resources that have been assigned to a role it plays.
Empowerment	We introduce the concept of Empowerment as a means of specifying the status changing rights of an agent. Empowerment is the right of an agent to perform status changing actions, such as "commit", "enable", "suspend", etc. Empowerment naturally falls into two classes: state and activity empowerment. State empowerment specifies the range of status through which an agent may take a state by performing the appropriate actions, such as commit. State empowerment not only specifies allowable status changes but may be used to restrict the set of resources an agent is empowered to commit to a use/consume state. An agent may be empowered for any type of resource, including other agents. The implication being the first agent may commit the second to a state.

3.6.3 Resource Description Framework

The Resource Description Framework (RDF[33]) is a family of W3C consortium specifications for describing the relations between web-formats. As it has been adopted in industry and by the community in general, further uses have been used for describing general conceptual modelling.

A more complete description is included in Deliverable D4.5 and to avoid repetition only the elements concerning Dependencies will be described in this Deliverable.

[33]http://www.w3.org/RDF/

Overview of the framework

RDF is a model that was developed to allow for data interchange on the web. RDF extends the linking structure of the web to allow for the description of relations of data resources by linking two objects with a relationship. To this end, RDF provides a formal syntax and semantics for so-called triples. Each triple consists of a “subject”, a “predicate” and an “object”, usually denoted as “(subject predicate object)”. A set of one to many RDF triples is referred to as an “RDF knowledge-base”.

Approach taken

The semantics used varies between the different nomenclatures used but generally a serialised description of triples are used to describe two entities and their connected relation. The order of the two objects is important as there are some uni-directional properties of relationships. If a relationship is symmetrical then the order is not important. The approach ends up intrinsically deriving the description of a multi-directed, labelled graph.

Dependency relations taken into consideration

At the base any relationship can be defined between two entities. RDF does not stipulate the use of, or provide any base relationships. This means that there is a large amount of flexibility but it also means that all the rules for the relationships have to be defined for anything that wishes to make use of this construct. As the description is normally text-based it also makes it difficult without adding an extra indirection level for using weighted or valued dependency relations. Using the extensibility of XML and XSD schemas it is possible to extend these dependency relations.

Implementation examples

A simple example of RDF is copied from W3Schools[34]

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:cd="http://www.recshop.fake/cd#">

<rdf:Description rdf:about="http://www.recshop.fake/cd/Empire Burlesque">
        <cd:artist>Bob Dylan</cd:artist>
        <cd:country>USA</cd:country>
        <cd:company>Columbia</cd:company>
        <cd:price>10.90</cd:price>
        <cd:year>1985</cd:year>
</rdf:Description>
```

[34]http://www.w3schools.com/rdf/rdf_example.asp

```
<rdf:Description rdf:about="http://www.recshop.fake/cd/Hide your heart">
        <cd:artist>Bonnie Tyler</cd:artist>
        <cd:country>UK</cd:country>
        <cd:company>CBS Records</cd:company>
        <cd:price>9.90</cd:price>
        <cd:year>1988</cd:year>
</rdf:Description>
</rdf:RDF>
```

This example uses two name-spaces, one for “rdf” and one for “cd”. Each namespace is further defined using the schema that are linked to at the beginning of the description. This example is of RDF in XML serialised fashion. Each of the relations are those denoted by the properties (eg. Cd:country, relation is country). The object of each relation is the value stored between the markers (eg. For cd:country for Hide your heart is UK). This means that the subject : relation : object would be, cd/Hide your heart : country : UK.

Limitations

Most of the limitations are due to the serialisation of RDF when used in RDF/XML form. The relationships in larger examples soon becomes unwieldy and is complex to parse with traditional XML parsers. Apart from that the main limitation is that since RDF is so flexible it requires work before it is usable. More discussion into the limitations of RDF can be found on the web[35].

3.6.4 Web Ontology Language RDF/OWL

The Web Ontology Language (OWL) is a family of knowledge representation languages for authoring ontologies. RDF is flexible enough to serve as a basis for development of OWL and as such is one of the more common serialisation formats for OWL.

A more complete description is included in Deliverable D4.5 and to avoid repetition only the elements concerning Dependencies will be described in this Deliverable.

Overview of the system

The OWL standards in it’s different versions are consecutive approaches from the Semantic Web community which are based on a small subset of the family of Description Logics (DL). Each DL is a restricted part of predicate logic, whereby all description logics have in common that they are decidable (which predicate logic is not) to make them practically tractable.

[35]http://eulergui.sourceforge.net/n3_rules_good_practices.html#L587

OWL-DL (Description Logic) is preferred to be used due to the possibility to reason about it unlike the highly expressive full variant. More detail is given in D4.5.

Dependency relations taken into consideration

The dependency relations in OWL are more expressive than raw RDF which has no formal relation mappings. Based on description logic subsumed into OWL there are a series of relations that help to explain whether an entity is in the same range and domain of others as well as other relations. More recent additions have allowed further expressibility for properties in OWL. “Number Restrictions”, “Inverse-Roles” and “Role Inclusions” are examples of some the formal extensions implemented by the latest version of OWL.

3.7 Digital Preservation

The last aspect to be discussed in this section is digital preservation that is relatively novel to the enterprise community and is being addressed by TIMBUS. The modelling of dependencies of digital objects is one of the primary concerns of digital preservation. It is required to keep objects and their dependencies understandable, accessible and unrelieved. The fact that the context of an object changes over time adds special requirements to digital archives. In this part we reflect the common understanding of information modelling in an archive and projects dealing with changing contexts and scopes.

3.7.1 OAIS Information Model

The Reference Model for an Open Archival Information System (OAIS) [OAIS, 2002] is the de-facto reference model for digital preservation. It has also been adopted by ISO as standard ISO 14721:2003 [ISO14721:2003]. It defines the entities and relationships of the digital preservation domain, providing a common terminology that can be used when approaching the problem of building archival systems that are capable of effectively performing digital preservation.

Besides providing the terminology, it also provides a reference information model for guiding the implementation of information packages for preservation. The OAIS considers that an Information Object is composed of a Data Object and the Representation Information, which adds meaning to the data object, so that it can be interpreted in the future. The Representation Information might contain Structure Information, which describes the way the data on a data object is structured, Semantic Information, which provides meaning to the structures defined by the Structure Information, and other Representation Information, such as Representation Networks,

which might contain all the linkages of Data Objects and Representation Information required for interpreting a Data Object[36].

In order to be preserved, information needs to be packaged along with the information that allows its interpretation in the future. Three types of Packages are defined in the OAIS: the Submission Information Package, for submission by the producers of information into the archive for preservation; the Archival Information Package, for long term preservation; and the Dissemination Information Package, for dissemination into the future consumers of information.

Different specialisations of Information Object are possible:

- *Content Information*, which represents the data object targeted for preservation and the accompanying Representation Information;
- *Preservation Description Information*, which includes information that is needed in order to adequately preserve the Content Information;
- *Packaging Information*, which binds or related the components of the package to be preserved (Content Information plus Preservation Description Information) into an identifiable entity;
- *Descriptive Information*, which allows the search for and retrieval of the information packages.

The main classes and relationships within the OAIS information model are depicted in the UML-class diagram in Figure 9.

It's shown, that the OAIS offers possibilities, to preserve the relation between objects and at least parts of the semantic of data.

[36]Representation Information is itself a Data Object, and as such it might need its own Representation Information

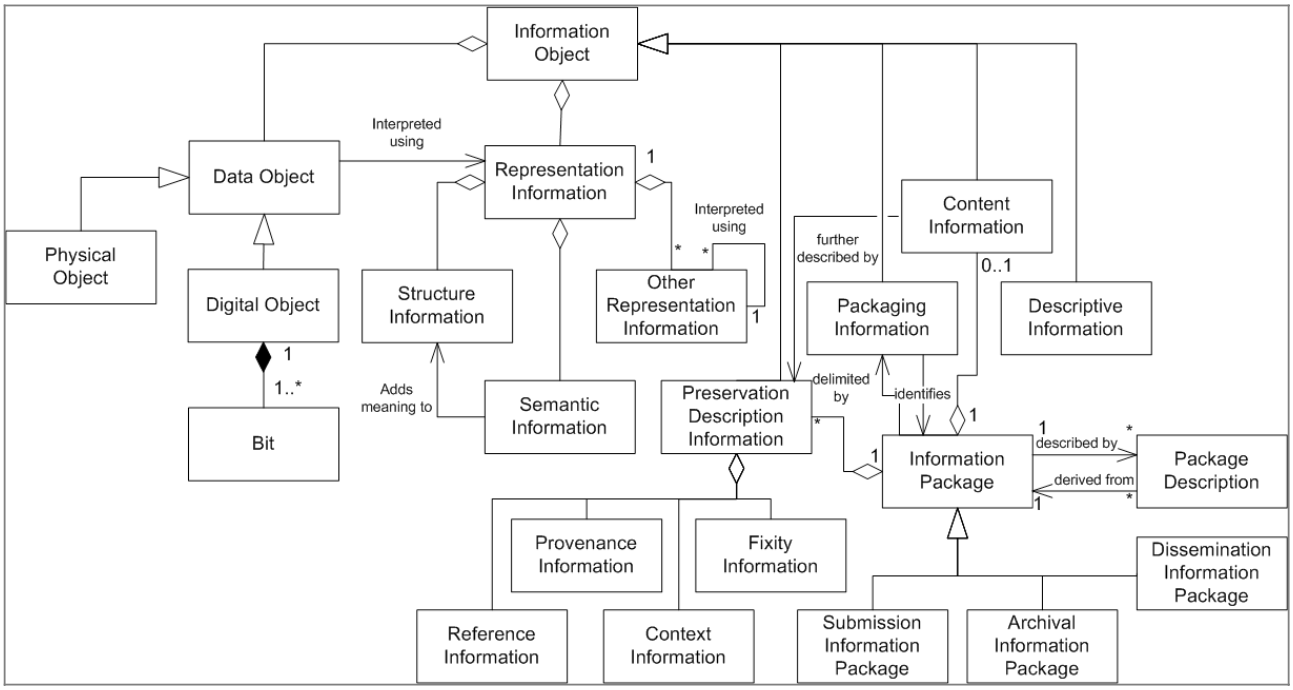

Figure 9: OAIS Information Model [OAIS, 2002]

3.7.2 PREMIS

The PREservation Metadata: Implementation Strategies (PREMIS[37]) Working Group has defined a data dictionary of preservation metadata relying upon the concepts of Intellectual Entity, Object, etc, and on the relationships between these conceptual entities. The PREMIS preservation dictionary [OCLC, 2005] is implementation independent as the elements define information needed for preservation regardless of how that information is stored. By means of the preservation dictionary, PREMIS provides semantics for digital archives (and therefore, for OAIS). The semantics from PREMIS carry dependency relations for information such as usage notes, applicability, object categories, data constraints and a rationale.

An Intellectual Entity is a set of contents that can be described as a unit. For example, an Intellectual Entity as a web site may be composed of many objects, such as several web pages and images. An Object is the basic unit of information in the digital form. object entities contain preservation information about the object which it refers to, such as checksum, digital signature, provenance information, and relation to other objects, e.g., the source object to the migration.

[37]http://www.loc.gov/standards/premis/

3.7.3 CASPAR Preservation Networks

CASPAR - Cultural, Artistic and Scientific knowledge for Preservation, Access and Retrieval - is an Integrated Project co-financed by the European Union within the Sixth Framework Programme (Priority IST-2005-2.5.10, "Access to and preservation of cultural and scientific resources")[38]. The aim of the project is to fulfil the need of maintaining understandability over the long-term [Giaretta, D., 2007].

CASPAR developed the notion of preservation network models with the intention of representing digital objects and relationships, depicting the dependencies existing between objects, so that these can be understood in the future and preservation objectives are met [Conway, E., et al., 2011]. These preservation networks can then be stored in registry repositories of representation information, so that knowledge can be reused.

Preservation networks are represented in a similar fashion to that of class diagrams, depicting to kinds of entities: Objects, which are uniquely identified digital entities with the attributes of information, location, and physical state; and Relationships, which have the attributes of function (for depicting any necessary function to be performed on object), risks and dependencies, tolerance (if the absence of a determined function is critical or not), and quality assurance and testing (if a determined function has been subjected to testing or quality assurance). Relationships can be composed in to alternate or composite relationships, depicting respectively the fact that only one relationship needs to function or the fact that all the relationships must function in order to fulfil the objective. The graph in Figure 10 depicts such a preservation network.

[38]http://casparpreserves.eu

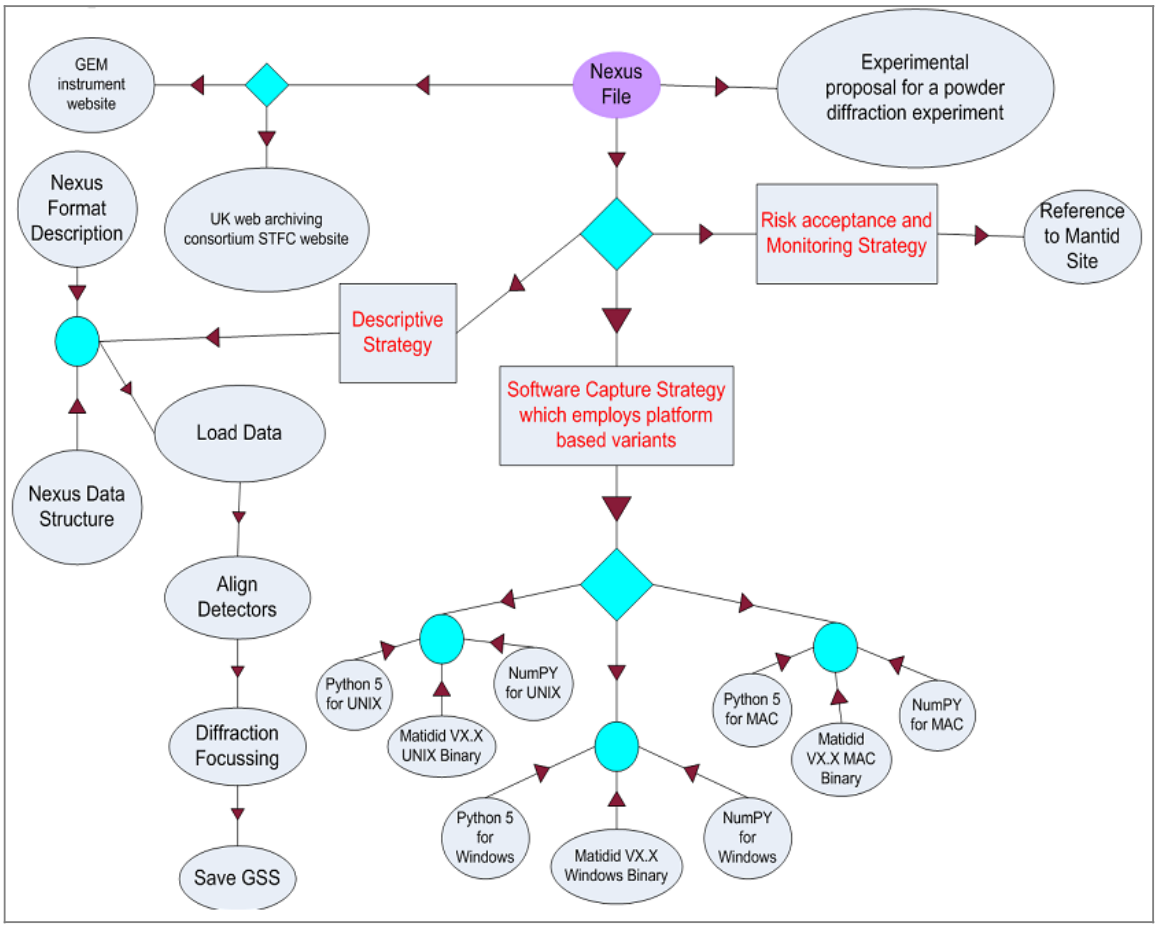

Figure 10: Preservation Network Example [Conway, E., et al., 2011]

Preservation actions can be enforced on networks, which can alter their structure. Different kinds of actions can be applied on preservation networks: Risk acceptance and monitoring, which involves the active monitoring of dependencies and the acceptance of risks in case an object is dependent on external information.

CASPAR Preservation Networks describe objects, classes and relations relevant to digital preservation of artefacts whereas the Formalism describes the relevant objects, classes and relations for digital preservation of business processes. The distinction between digital artefacts and business processes separates the work between TIMBUS and CASPAR. Another key differentiating factor between the two projects is that the purpose of the Formalism is to capture the formal semantics to enable reasoning mechanisms to work with the model generated. This does not simplify the problem but allows for the problem to represented in a manner that allows for automated, correct and complete reasoning tools to be used that exist within the knowledge engineering community.

4 Formal Language Specification

This Deliverable presents the first iteration of a formal language for describing dependencies throughout the scope of an enterprise. Due to the fact that dependencies link classes of entities and that the contextual information is based around the entities, there is a strong link between the work carried out in this Deliverable and that of D4.5 from the first Context Modelling Deliverable (Business Process Contexts), from Task 4.4. To avoid repetition in the description of the Formalism from this Deliverable we focus on the categorisation and description of dependencies whereas D4.5 investigates further into contextual parameters and properties of entities as does the subsequent Deliverable, D4.9 (Refined Business Process Contexts).

The first iteration of the Model proposed and discussed in this Deliverable will be further detailed in the subsequent Deliverable based on the same Task (T4.2), D4.3 (Dependency Models Iter. 2). This refinement will look to further improve the description of entities and better express the relationships between components such that further reasoning can be applied to the problem space. The model is a representation of the connectivity between various components specified in the Formalism.

For the purpose of modelling the dependencies (including semantics) existing between the entities that compose a business process and its context, different knowledge representation mechanisms can be used. However, there is a specific requirement for enabling reasoning on top of this knowledge in order to identify relevant dependent context components for preservation purposes, which will be explored in D4.3. This makes the usage of Ontologies appropriate for capturing this knowledge, since automated correct and complete reasoning mechanisms can be used to solve problems that otherwise would be too complex to solve by humans.

In [Steffen S., et al., 2009], ontologies are defined: “An ontology is a formal, explicit specification of a shared conceptualisation for a domain of interest.“ This definition focuses on the important characteristics of an ontology and in turn is a refinement of Gruber's definition “An ontology is a specification of a conceptualisation” [Gruber T. 1992]. An ontology specifies an abstract model of a domain of our world, also referred to as the “domain of discourse“. The model formally defines all concepts and their relationships which are relevant in the domain of discourse.

Depending on the scenario, the complexity of ontologies may vary. The expressiveness of ontologies ranges from basic taxonomies to complex networks of concepts, relationships and rules on these concepts and relationships. The focus in TIMBUS and specifically in this Deliverable, D4.2, is to denote the information specific

to dependencies within the backdrop of the context model developed in D4.5. The domain we are interested in analysing is broad and complex, a generic enterprise. Enterprises come in many shapes and forms and as such the parameters that we wish to investigate have to be clearly defined to try and reduce the scope and complexity of a possible model. From the motivation in Section 2.2, the Formalism has been designed to model the parameters in an enterprise relevant to answering the question "What elements need to be captured in order to digitally preserve a required business process."

4.1 Base ontology and construction

The Formalism is a description of the syntax and rules used for describing, naming and providing the rules that all elements that are represented by it, must adhere to. Elements in this case refer to all the possible components that a business process could comprise. When captured in the model, elements will be described as entities. For TIMBUS a few representational schemes were analysed and OWL was chosen as the representation format. The tool used for developing the Formalism is Protégé[39] from Stanford University that is a well established tool for developing ontologies. OWL provides a base-structure for the elements to be represented as entities, allowing for the creation of entities and denoting the relationships between entities. Entities can be further sub-divided in to classes and instances (known in Protégé as individuals). Classes are used to describe generic types of instances. Classes can be described as an abstraction that contain properties which all members of the class adhere to. Deciding if an instance should be a further sub-category of an existing category or an individual instance is decided when the generic model is designed[40]. Instances can have further properties as specified by data properties (using Protégé terminology). Relations between entities are the descriptive term used to link two or more entities. For OWL, the modelling technique only allows for binary relations (between two entities) and so when relations are discussed they refer to directed (order of the description of the entities is significant), binary relations.

The generic term model is used to denote the structure that entities and relations can have as represented by the Formalism and is an abstraction of the real world. The base model includes all the base entities and relations that have been created for capturing a business process. An instantiated model is a model that has been populated with the entities that represent the elements of a real business process for a specific scenario. Throughout the description the instantiated model will be referred to more often as it is the object that will be used for describing the specific business

[39] http://protege.stanford.edu/download/download.html

[40] http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html

process scenario that has been captured and as such instantiated model will be abbreviated to model. Where the generic model is referred to it will be labelled as the generic model.

The main design and construction of the Formalism was carried out as work for both Tasks T4.2 and T4.4 (capture of contextual information and metadata) in TIMBUS. The relations between entities and the entities themselves are interlinked and so it is difficult to separate the concerns to purely dependency or purely contextual information. The design of the contextual parameters was developed mostly as a result of Task T4.4 and as such there is some information that will be common to both Deliverables. To avoid repetition of the work that is explained in that Deliverable, only a brief discussion about the decisions taken when developing the contextual parameters will be presented in D4.2. More detail and explanation will be provided in terms of the relations and dependencies within Task T4.2 and this Deliverable D4.2.

As can be seen in a representational snapshot of the ontology as shown in Figure 11, all the entities in an enterprise will be sub-elements of a root class, 'Thing'. All classes of the ontology are derived from a generic super-class, 'Thing', through sub-classes. The entities in the upper half of the diagram above the asserted knowledge line are classes and sub-classes. The entities in the lower half, in the asserted knowledge section of the diagram shown in Figure 11 are individual instances of the classes. In TIMBUS the instances will relate to individual elements in the system and their attributes. A subset of these instances will be selected for preservation of a specific business process. The classes on the other hand have been developed to attempt to group the concerns of different aspects of enterprises. These relate strongly to the Domain Specific Languages (DSLs) referred to in the related work (Section 3), where the languages have been defined to categorise similar instances and define specific properties about them.

For instance one of the sub-classes has been defined to be software. This is following the identification in many scenarios related to the TIMBUS context workshop (held in relation to D4.5) where partner specific scenarios were developed that software is a class that is relevant to many of the scenarios. There is however no one correct way for deciding the hierarchy of classes and for deciding the depth of detail that a model should capture and at what point instances should be used and is in fact one of the limitations in using ontologies. As such the approach is to take a middle-out approach as suggested in Uschold and Gruningers' methodology [Uschold M., et al., 1996]. For this the middle-out approach uses a combination of top-down and bottom-up design to help design the structure of the class hierarchy.

The top-down approach is where, using some previous understanding and experience, the designers or knowledge workers enforce some level of categorisation, by assigning classes and sub-classes. The bottom-up approach however uses the assignment of

properties and attributes to individual instances that direct the categorisation efforts. A software reasoner attached to the model can use a combination of the definitions assigned to classes, the data properties and object properties to decide in which classes instances should belong. A software reasoner can also assist in determining if there are equivalence classes, where two classes denote the same set of properties and highlight an error or the need for more detail to be applied. Some of this automatic categorisation might not match the intention of the developers of the model but then using refinements in the Formalism, the classes can be redefined.

Constructing “well-structured” and “well-named” hierarchies of classes and relations, and classifications of instances, is a complicated process. Ontology designers (also called knowledge engineers) have a subjective perspective in modelling the knowledge on a part of the world (their domain of interest). Their perspectives may differ on what are the relevant parts of the domain of interest, and in their perception on the suitable granularity of classes and relations in the hierarchy. This can easily lead to inconsistencies where classes, instances and relations should be placed in the hierarchies “to best model” the domain of interest. Firstly, naming conventions specify the proper naming of classes, instances, relations and dependencies in TIMBUS. Secondly, design pattern conventions establish best practices in modelling frequent problems, for example, “how to represent information objects, such as a text, and its information representations, i.e. its physical representations, such as a PDF or a printed book” and “how to establish a new class in the hierarchy”. The naming conventions used in the Formalism are briefly described in Section 4.2, but are provided in more detail in Deliverable D4.5.

From the snapshot of a part of the TIMBUS Music Process in Figure 11, information could be used that has been captured in this model. It may be used for determining if the business process is preservable when it is first being preserved. Information captured in the Formalism and the specific model could be used to determine if there are any problems in exhuming the system to a new environment. For example the software that is represented in the diagram is Taverna that requires a Java Virtual Machine to run. It may happen that due to company policy that at the time the business process is being exhumed and brought to a live-system that there is a specific problem in using the licence, GPL version 2.0. In this case the business process would need an alternative component to perform the role of the Java Virtual Machine. The specific Java Virtual Machine instance, which is named HotSpot JVM, is a particular Java Virtual Machine that uses that specific licence. In this case an alternative, JikesRVM that uses the Eclipse public licence could be a suitable alternative. If there are particular features that a Java Virtual Machine must have for it to run the Taverna software then this would have to be encapsulated in the model and would lead to a refinement that is more descriptive that allows for this dependency to be captured.

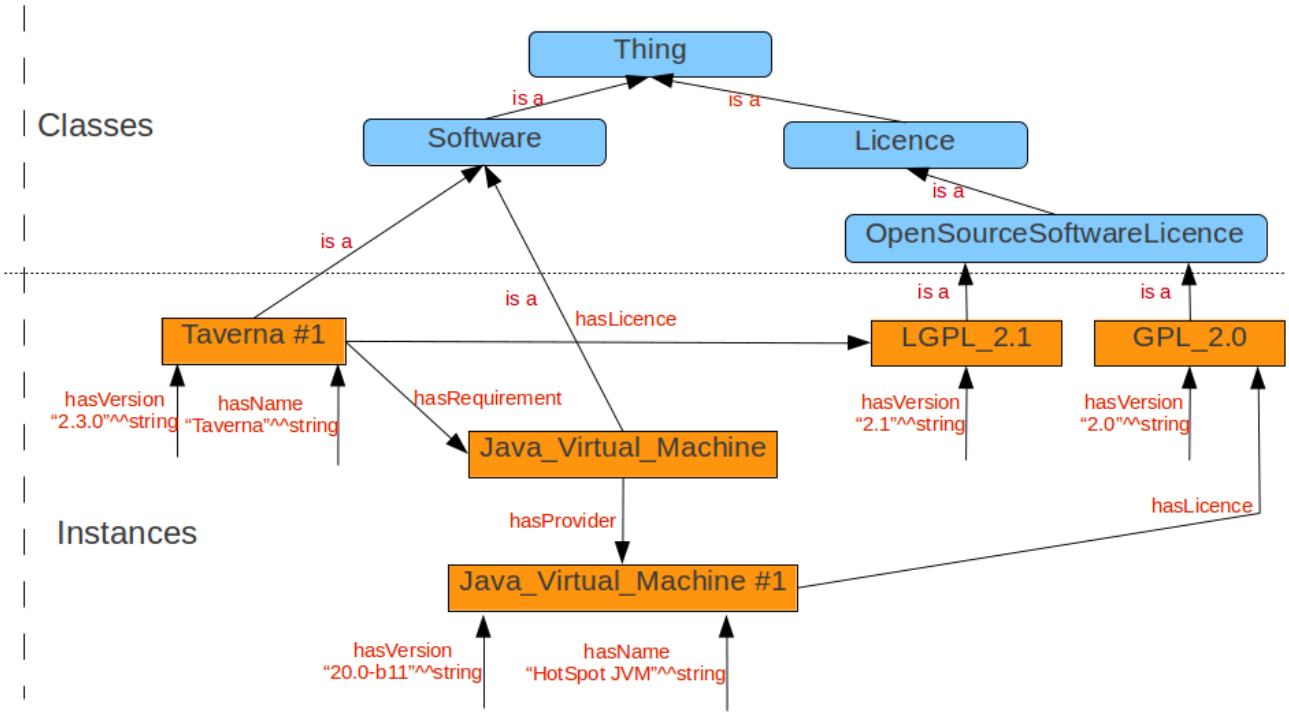

Figure 11: Representation of classes, instances and relations in the ontology

4.2 Naming conventions

In TIMBUS, each word contained in a term starts with a upper-case first letter and continues with lower-case letters. For terms consisting of a sequence of words the camel-caps notation is to be applied, for example: “SportsEquipment”.

With ontology formalisms like OWL, usually each ontology consists of only one global name space. To identify an element of an ontology, the name of each element has to be unique. Each class name, individual name and relation name has to be unique in the Formalism in TIMBUS. To prevent naming conflicts the following guidelines have been established for naming classes, individuals and relations:

Each class name is prefixed by “c_”, indicating that this name belongs to a class. In TIMBUS, the fully qualified class name from the class hierarchy is used to differentiate classes that otherwise could reside in two or more classes and to specify them. As parent classes follow this naming convention, this is sufficient for disambiguating class names.

Each individual name is prefixed by “i_”, indicating that this name belongs to an individual relation, and to disambiguate it from classes and relations. Instances may

be named the same and are differentiated by the minimal set of descriptive relations required to disambiguate all the individuals in the class. Syntactically, the suffix for each relation after an individual's name is constructed by concatenating two underscores (“__”), the relation's name (e.g. “hasVersion”), one underscores (“_”) and the related individual's name (e.g. “V10.0”).

Each relation name is prefixed by “r_”, indicating that this name belongs to a relation, and to disambiguate it from classes and individuals. In contrast to classes and individuals, in TIMBUS, the first word contained in a term starts with a lower-case first letter and continues with lower-case letters.

For sake of brevity in this Deliverable as the terms are being described the naming convention will not be held, but in any implementation the naming conventions should be used to avoid the problem of naming conflicts.

4.3 Formal language for modelling of dependencies

The intention was to start modelling dependencies based on a previous work carried out for a previous project (MANCOOSI[41]) that described formally the dependencies between components from a software and packaging perspective [Mancinelli F, 2006]. The Common Upgrade Description Format (CUDF[42]) is described in more detail in Section 3.3.2. This description format is the basis for the work on some of the more general dependencies descriptions in Annex (Annex A.1 - Fundamental Concepts). As defined before dependencies are relations between entities. An example of how the Formalism developed in TIMBUS can be applied is that of the TIMBUS Music Process that will be described in Section 5, but is basically a representative, technical use-case scenario that has been developed within the TIMBUS project to exercise the complete preservation process.

An example of how the TIMBUS Music Process would be represented if captured in CUDF is included as an Annex (Annex A.4 - Example Listing of CUDF for the TIMBUS Music Process in Taverna). CUDF was designed for capturing the domain of software dependencies on GNU/Linux Operating Systems. As such CUDF has limitations in that the expressiveness of that particular description format is limited to describing the existence of software packages on a single GNU/Linux Operating System and the relations between those software packages if stipulated by the person that packaged the software and the packaging software. The relations captured between packages are normally strict constraints that help guide SAT-solvers (satisfiability) to determine if software can be upgraded on a system as explained in Section 3.3.2. CUDF therefore is limited in what type of information it can express and a more descriptive system

[41] www.mancoosi.org

[42] http://mancoosi.org/reports/tr3.pdf

was required. This Formalism therefore was designed to attempt to capture components of a business process and the relations between those components requires more expressiveness to relate components that may not naturally have any connections. Software dependencies relate packages of particular versions together and state whether packages are needed on the same system or if they cannot be present on the system at the same time (a conflict). CUDF however cannot express cardinality, which means it cannot work with the countable number of entities in a system. For example if a system requires to have four people working on it for the process to be valid, there is no way of representing this information using CUDF without creating an entity that is specifically 'four people'. As such the expressiveness of CUDF is too limited for the envisaged scenarios without extending the original language and the connected solvers.

The specific language that is used for defining the Formalism at the base level is OWL2 on RDF. This is a combination that seems to work well and is similar in many ways to RDFS (RDF Schema). The use of OWL is an approach that is widely used in industry and there is a large community that has developed around creating ontologies using OWL has a number of associated tools for working on design and using the ontologies. To maintain extensibility it is then thought that elements will be defined using DSL terms from the related work in Section 3. As stated in the introduction in Section 2, reasoning can be used within the ontology to determine if all the relations are met for determining whether a business process can be preserved and that all the required constraints have been met. The use of external solvers has not yet been ruled out for determining sub-problems and would make the process of solving the entire problem set simpler in that part of the problems can be solved using pre-existing tools. However any knowledge that is not represented within the base ontology cannot be reasoned upon, so either a translation stage would be required or there would need to be integration performed to allow for the results to be used in a meaningful way.

4.4 Types of dependency relationships

There will be two classes of dependency relations that will be formally defined and represented in this Formalism. One set of relations will be that of constraint relations that are strict relations that must be met in order for the process as captured through the model of the system, to be preserved. Constraint relations therefore will be relations between elements that must be adhered to for the potential solution for the specific model to be valid. Constraint relations are critical relations that the reasoner must solve. The other set of relations is a weaker set of descriptive properties that associate entities with attribute details otherwise known as description relations. Given the description relations of the entities and the other instances captured in the system and the context, other constraint relations may be inferred. Using a combination of the constraint relations and the description relations it should be

possible for the software reasoning tools to solve what entities must be captured in order for preservation to succeed, or otherwise suggest that using the current model of the process that it is un-preservable and display the constraints leading to that conclusion. Then using a combination of information provided by process experts through the tools that will be generated in WP6 (Intelligent Tools and Technologies to Support Digital Preservation of Business Processes), the constrictive relations may be relaxed in order to make the system preservable (relaxing the constraints).

The constraint relations that will be used are captured in Table 3 and are related to the generic constraint relations for software dependencies mentioned in Section 3.3.2, in Table 2. The descriptive relations that will be captured are a non-exhaustive list represented in Table 4. These relations have been captured as a result of analysing partner specific scenarios at context workshops related to D4.5 (Business Process Contexts) and the domain specific terms presented in the related work, Section 3.

The relationships defined are mono-transitive relationships. This means that each subject takes a single object. There are more complicated relationships that require two or more relations but these are not easily representable in OWL-RDF that can only represent binary relations but can be manipulated to represent n-ary relationships[43]. This is unlike UML that can have multiple arguments.

4.5 TIMBUS constraint relationships

The constraint dependencies in Table 3 are constrictive, generic dependencies that can be used to describe relations between entities at any layer in the Enterprise. The constraint dependencies could be incorporated into a knowledge-base or a static resource to dependency mapping. A resource to dependency mapping would be a model whereby each element has a constraint relationship to other elements that are necessary and lacks the expressiveness to detail weaker relationships between elements in a business process. The descriptive relations allow for weaker connections to be associated between context parameters and domain that the enterprise is in, to allow for the reasoner to calculate more connections. This is important for digital preservation because the complex question that we are attempting to answer is if the model that we generate of a business process is preservable. If all the relations in the system are already denoted with constraints by humans then the solution would be to traverse the dependency graph and check that all the required elements are in the solution set.

[43] http://www.w3.org/TR/swbp-n-aryRelations/

Table 3: TIMBUS constraint relations between entities

Name of relationship	Example	Meaning
hasRequirement	Taverna-workbench hasRequirement JavaVM	Entity1 can not exist in the system without Entity2 existing at the same time.
hasConflict	Taverna-workbench hasConflict JavaVM(Version==1.7)	Entity1 or Entity2 can exist on the system but cannot exist at the same time. Entity1 and Entity2 are mutually exclusive.
relationBefore	PurchaseService pre-depends(time) RESTfulProvider	Entity1 requires that Entity2 be in the system at least for time before it can exist in the same system. Entails an ordering of dependencies.
relationSameTimeAs	EnterpriseCrediting same-depends CustomerDebiting	Entity1 requires that Entity2 be activated on the system at exactly the same time. Ordering
relationAfter	NetworkConnection post-depends(time) NetworkThroughput	Entity1 requires that Entity2 only be in the system at least for time after Entity1 has been activated on the system. Ordering

4.6 TIMBUS descriptive relationships

The descriptive dependencies or attributes shown in Table 4 however are more generic relationships that can relate entities using more natural language. The relations are more reliant on the perspective that they are trying to describe and as such are more applicable to reasoning methods. Expanding upon the constraint relations and by using more intuitive natural language relations it is easier to relate entities without having a human decide whether a relation is important or not. This way it is easier to map real-world situations into the Formalism for reasoning as a human does not have to decide if it is imperative that two entities be connected. It does however make the reasoning more complicated as the relations are less constrictive and a set of rules have to be derived that allow for the reasoning to be domain specific to the type of enterprise and the components available in its context.

Table 4: TIMBUS descriptive relations between entities

Name of relationship	Example	Meaning
hasAssociation	Permission hasAssociation Resource	Entity1 has a generic association with Entity2.
hasAuthorisation	Person hasAuthorisation RunSoftware	Entity1 has permissions as granted by Entity2.
hasComplement	BusinessGoal hasComplement Person	Entity1 is partly assisted to completion of associated process by Entity2.
hasCompetency	Person hasCompetency SoftwareDevelopment	Entity1 has a skill for completing certain processes as denoted by Entity2.
hasConfiguration	JavaVM hasConfiguration VMreqs	Entity1 is configured in a certain manner as defined by Entity2.
hasCreator	SoftwarePatent hasCreator Person	Entity1 has been created as a result of something that Entity2 has performed.
hasDate	ServiceDataRetention hasDate RenewalDate	Entity1 is associated with a date as specified by Entity2.
hasDeliverer	Email hasDeliverer Email-Server	Entity1 is associated with a delivery mechanism Entity2. Can be connected with hasRecipient to denote target of Entity1.
hasEncryption	HTTPS hasEncryption SSL/TLS	Entity1 uses an encryption scheme as defined by Entity2.
hasExecution	Taverna-workbench hasExecution MusicClassification	Entity1 runs the process associated with Entity2.
hasExecutionEnvironment	Taverna-workbench hasExecutionEnvironment JavaVM	Entity1 uses a working environment Entity2 to be performed within.

Name of relationship	Example	Meaning
hasExecutor	ProcessPreservation hasExecutor PreservationAgent	The process as specified by Entity1 is performed by Entity2.
hasFormat	TextFile hasFormat Word97	Entity1 has an arrangement as specified by Entity2.
hasGuide	DataEncryption hasGuide SAS70Compliance	Entity2 is used to guide Entity1 in how the associated process should be executed.
hasImplementation	Cache hasImplementation FIFO-Buffer	Entity1 may have many ways to be implemented and Entity2 is used to specify a single implementation.
hasLicence	Taverna-workbench hasLicence LGPL_2.1	Entity1 has a licence as specified by Entity2.
hasLikeness	Permission hasLikeness Right	Entity1 is somewhat similar in function and purpose to Entity2. Used mainly to tag that more specific relations should be used for reasoning.
hasLimitation	Windows7Starter hasLimitation MemoryLimitation[44]	Entity1 has a limitation as described in Entity2.
hasLocation	Office1 hasLocation Lisbon-PT	Entity1 is located in a situation as specified by Entity2.
hasMilestone	Project hasMarker Milestone6	Entity1 has a distinctive point in its life-cycle denoted by Entity2.
hasModifier	OutputData hasModifier Software	Entity1 is modified in some way by Entity2. How it is modified depends on the process.
hasName	OS1 hasName Ubuntu-	Entity1 is named by a string. This is the name that is used

[44]http://msdn.microsoft.com/en-us/library/windows/desktop/aa366778%28v=vs.85%29.aspx

Name of relationship	Example	Meaning
	Linux	for querying rather than the name of the entity itself.
hasObjective	Division1 hasObjective CompleteProject	Entity1 has a high-level business goal as specified by Entity2.
hasObligation	Doctor hasObligation DutyOfCare	Entity1 has a duty to complete its actions under the boundaries of Entity2. It differs to rules in that obligations are dependent on societal environment.
hasOwner	Taverna-workbench hasOwner Person	Entity1 is owned in terms of property by Entity2.
hasPart	Enterprise hasPart Department	Entity1 comprises a component Entity2. Used to break down large components into smaller units without being the isA relationship. For example a phone hasPart antenna but an antenna is not a phone in itself.
hasPermission	Pilot hasPermission FlyPlane	Entity1 has a set of permissions as granted by Entity2.
hasProvider(weighting)	BusinessDocumentation hasProvider(weighting) Timesheet	Entity2 provides a functionality and a weighting of as to how good a solution it might be for meeting the needs of Entity1.
hasRecipient	Email hasRecipient Person	Entity1 has a target Entity2 for a process.
hasRecommendation(weighting)	BusinessProcess recommends BusinessDocumentation	Entity1 recommends Entity2 and suggests a weighting of as to how important it might be. There is no strong requirement that Entity2 be on the system. However if possible the solution set will try to include Entity2.
hasRedundancy	ServerFarm1 hasRedundancy	The process performed by Entity1 can be performed by Entity2.

Name of relationship	Example	Meaning
	ServerFarm2	
hasResponsible	Department hasResponsible Manager	Used to differentiate between hasOwner. All processes underlying Entity1 are under the responsibility of Entity2 unless a lower element is defined to have a different Entity responsible.
hasRole	Person1 hasRole Manager	Entity1 is specified a role within the Enterprise as specified by Entity2.
hasRule	Actor hasRule ActorRule	Entity2 specifies a set of rules that Entity1 has to comply to.
hasSnapshot	VM-1 hasSnapshot Snapshot-2	Entity1 is captured in its current state by Entity2.
hasSpecification	ARFF hasSpecification http://www.cs.waikato.ac.nz/ml/weka/arff.html	Entity1 is defined through documentation or a more formal schema that can be interpreted by a computer.
hasState	CPU1 hasState S3	Entity1 is described as having a state as defined by Entity2.
hasSupporter	Manager hasSupporter Assistant	Entity1 is supported for completing one or more processes it is involved with by Entity2.
hasThroughput	Server1 hasThroughput NetworkThroughput	Entity1 has a quantifiable output as specified by Entity2.
hasVendor	Computer1 hasVendor HP	Entity1 is sold by Entity2.
hasVersion	JavaVM hasVersion 20.0-b11	Entity1 has a version as specified by the value.
isA	OperatingSystem isA	Categorisation. Means that Entity2 is everything that Entity1 is, but may be more fully specified or

Name of relationship	Example	Meaning
	Software	have properties that make it less generic.

Figures 12 and 13 show how for the context parameters derived in D4.5, relations can be used to connect the various entities. Given n context parameters, if each parameter can have a single relation with another parameter the number of relations would be $O(n^2)$. From the context parameters there is not always a single relation that maps two parameters together, neither does it always make sense to have a connection between two unrelated parameters. As such the relations that have been derived from the existing context parameters are a non-exhaustive set that will be refined in the subsequent Deliverable, D4.3.

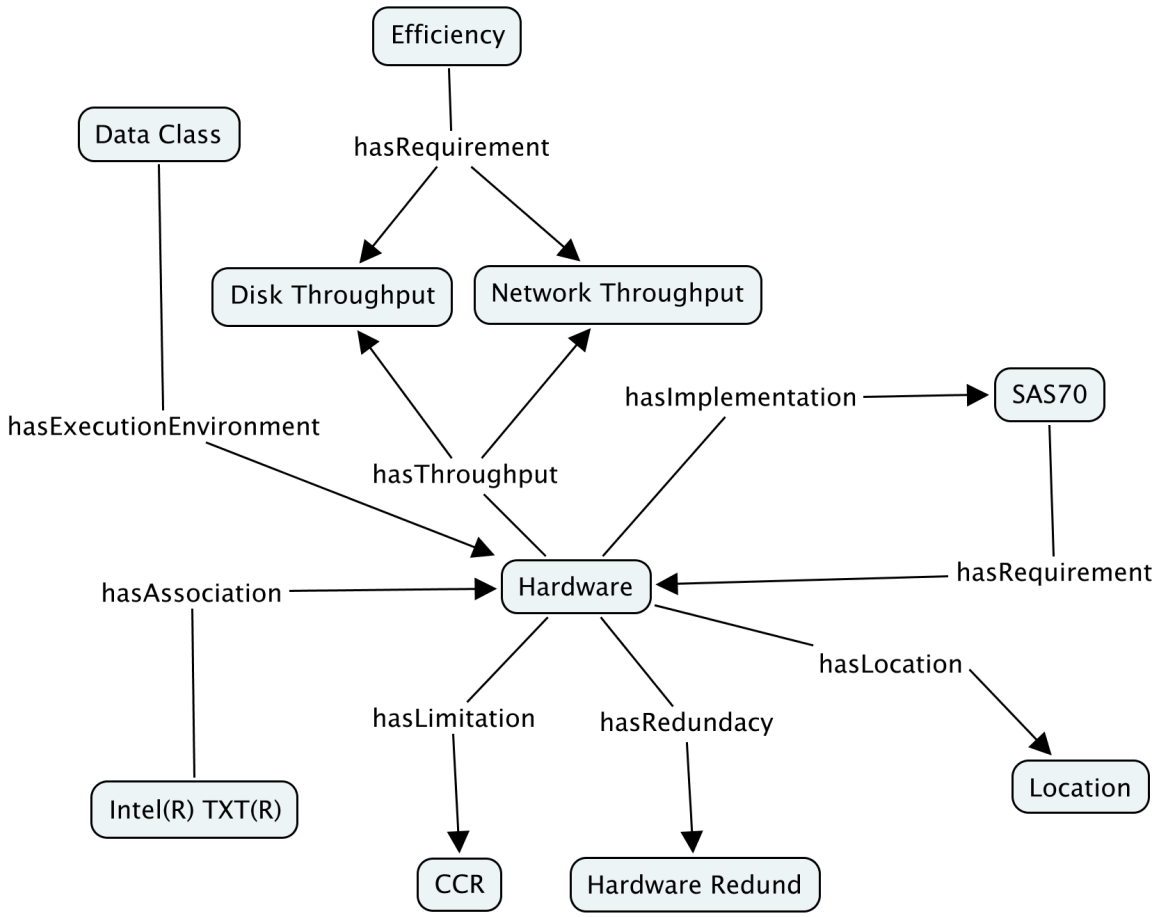

Figure 12: Demonstrative set of relations between context parameters in infrastructure layer

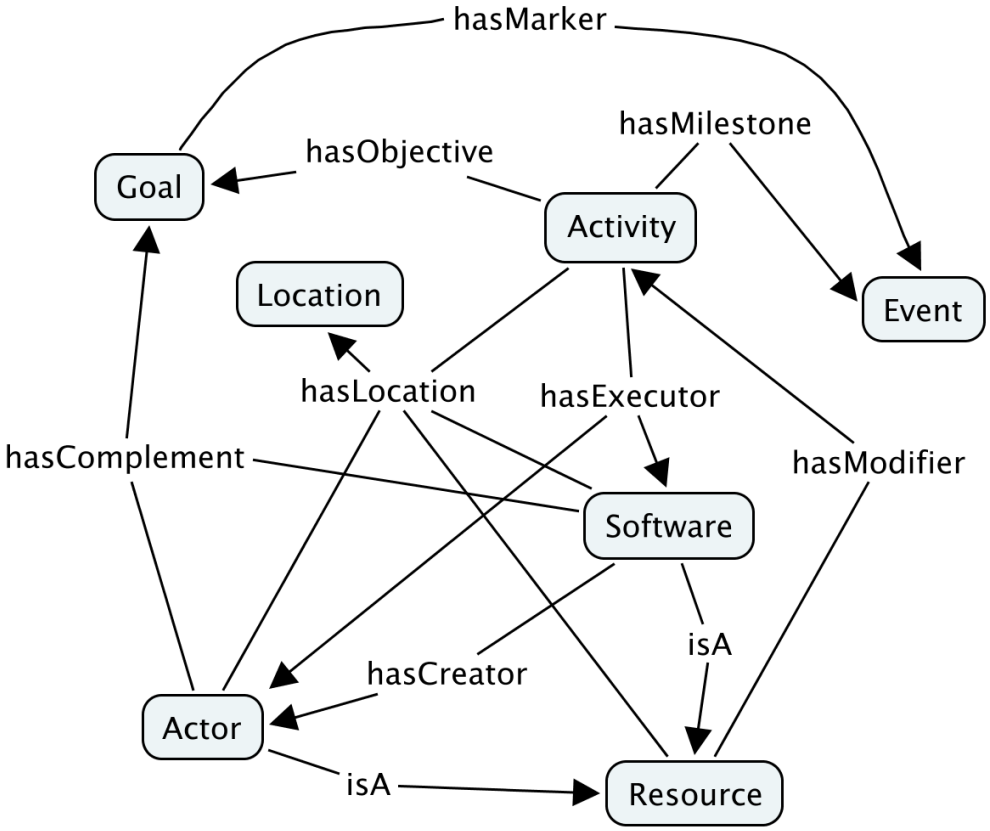

Figure 13: Demonstrative set of relations between context parameters in technology layer

4.7 Formal semantics for constraint relations

In this section we discuss the implementation of the constraint dependencies in Protégé. To do this the terms that are used are first discussed and then in Table 5 the data characteristics are presented. An Annex (Annex A.6 – Listing of OWL-RDF properties of constraint relations) also includes a listing of how these relations are implemented in OWL-RDF.

- Functional. Means that the relation can only hold between two entities. It could not then be further applied from one of the entities to another.
- Inverse Functional. Generally not used as it makes the ontology non OWL-DL compliant.
- Transitive. A relation that is transitive means that if a relation holds between two elements and a third related element is introduced that the relation will

hold. If A is an ancestor of B and B is an ancestor of C, then as a transitive relation, A is also an ancestor of C.

- Symmetric. This means that a relation is true in both directions for a pair of elements.
- Asymmetric. A relation that exists between a pair of elements cannot be applied in the inverse direction.
- Reflexive. For properties that are reflexive, the relation can be applied to the element itself.
- Irreflexive. An element that has an irreflexive relation means that it cannot have the relation applied to itself.

The domain and range for all these relations is the entire scope of the problem set.

Table 5: TIMBUS constraint relations between entities

Relation	Functional	Inverse Functional	Transitive	Symmetric	Asymmetric	Reflexive	Irreflexive
Conflict				**X**			**X**
Provide				**X**			**X**
Recommend							**X**
Require					**X**		
Before							
SameTime							
After							

4.8 Versioning and location of the Formalism

The work presented in this deliverable concerning the Formalism will be followed-up in Deliverable 4.3, which is due in month 24 of the project. The follow-up will be a

refinement of the Formalism presented in this deliverable and will deal with its application to the Industrial Scenarios being explored in WP7, WP8 and WP9.

Additionally, the Context Model and the Context Model Instances are to be named, versioned and published (project internally) in a consistent manner. The following serves as the initial version:

http://www.timbusproject.net/ontologies/2012/04/ContextModel.owl

The Context Model is published on the project website, using the above URL as a way of downloading the Context Model. A revision to this version of the Context Model will be provided at the end of the release of the next, Deliverable D4.3.

5 Application of Formalism to a use-case

In this section we describe how the Formalism can be applied to a specific use-case. This allows for some of the practical benefits of the Formalism to be demonstrated. It also forms the basis for how the more complete use-cases that are being specified as part of the work of Workpackages WP7, WP8 and WP9, could be represented by the Formalism. By testing on a complete but simple, technology use-case we can investigate some of the conditions that will be checked for and start to test the coverage of the current version of the Formalism. We first outline the process that we will be applying to the Formalism, providing details of how the workflow is constructed and then we will discuss how the relationships can be captured between the various components of the system and suggest how we might positively benefit from the modelling of this process.

The following applied example is on TIMBUS' Music Process. It is an example that is more focused on the technical aspects of TIMBUS and is an open-source process throughout. It uses a process flow engine that is open-source, Taverna, and all the components that are relied for the correct execution of the process are all open-source too. This has the benefit that there are no license or other restrictions from stopping the partners from describing and using the process.

Taverna is an open source and domain-independent workflow management system and is used for designing and executing scientific workflows. It allows for many small, well-defined tools or scripts be joined together in a pipeline to be able to reproduce the problem execution in a controlled environment. Data flow between services are specified without so much emphasis on how the services are executed. Taverna allows the TIMBUS Music Process to be specified as a set of connected processes and flows of data originating from specified inputs through to the expected outputs.

The example process is assigning meta-data to items in a specific dataset, such as a classification label to a music collection present in digital audio formats. This process is chosen for several reasons. First, it is similar in its nature to the eScience setting in Work Package WP7. Further, the process is interesting as it involves several different, partly remotely located services and tools. Finally, we have access to all relevant parts of the process components, and it has been created by one of the partners within the TIMBUS Project and so is a pre-existing work.

When performing automatic genre classification of a music collection, a process typically consists of (some of) the following steps:

- Acquiring a set of training and test data, potentially from remote content providers, such as Amazon.com, 7digital, etc. Those might provide a web service or another specific protocol to acquire the data.
- Further, meta-data such as a genre label might be acquired from the same or a different data source (such as the All Music guide[45], Gracenote[46], or MusicBrainz[47]. This assigned might be user generated data (e.g. via tagging), and thus change over time (e.g. from alternative rock to a later-emerging sub-genre of grunge)
- Pre-processing of the input data, e.g. format conversion from MP3 to raw audio, selection of relevant parts of the data (middle segments in music, specific paragraphs of a text, etc.)
- Extraction of representative, numerical features from the input data. There is a plethora of remote services emerging, e.g. echonest.com. Tools for extraction might also be used offline on the same machine, but might come in different languages (C++, Java, Matlab), depending on different third-party libraries. Remote services might change frequently, and extract new types of representations, or extract existing representations in a different way, thus providing different numerical values that might significantly change the outcome.
  Local implementations might utilise some of the computation algorithms provided with the specific language, such as the Fourier transform by Matlab. These might differ over different languages.
- Storing of the features in some way (text, database) and format (e.g. WEKA ARFF)
- Using a machine learning toolkit to train a model and assign new meta-data (genre labels) to unknown data. Different versions might provide different implementations of algorithms, and can thus lead to different results.

This process is illustrated below in Figure 14:

[45]http://www.allmusic.com/

[46]http://www.gracenote.com/

[47]http://musicbrainz.org/

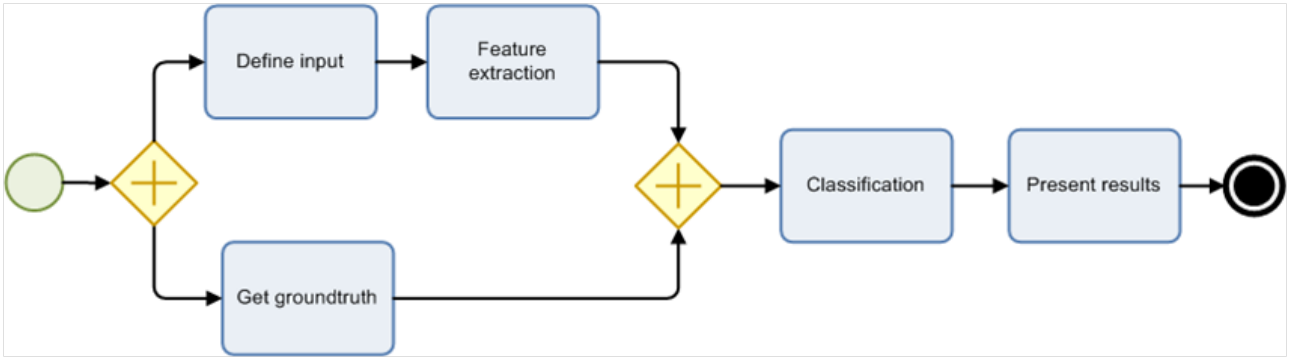

Figure 14: Dependency extraction can be used for determining boundaries of systems and Enterprise Processes

A specific instance of this process has also been modelled in the Taverna Workflow engine (http://www.taverna.org.uk/). A screenshot of the workflow can be seen in Figure 15. In the screenshot the workflow engine specific scripts are marked with 'Ws'; scripts based on predefined operations are marked with 'Ps', such as Base64 encoding.

Static input data is marked as 'Ip', while the process outputs are marked with 'Op'. The RESTful service is marked with 'RS'.

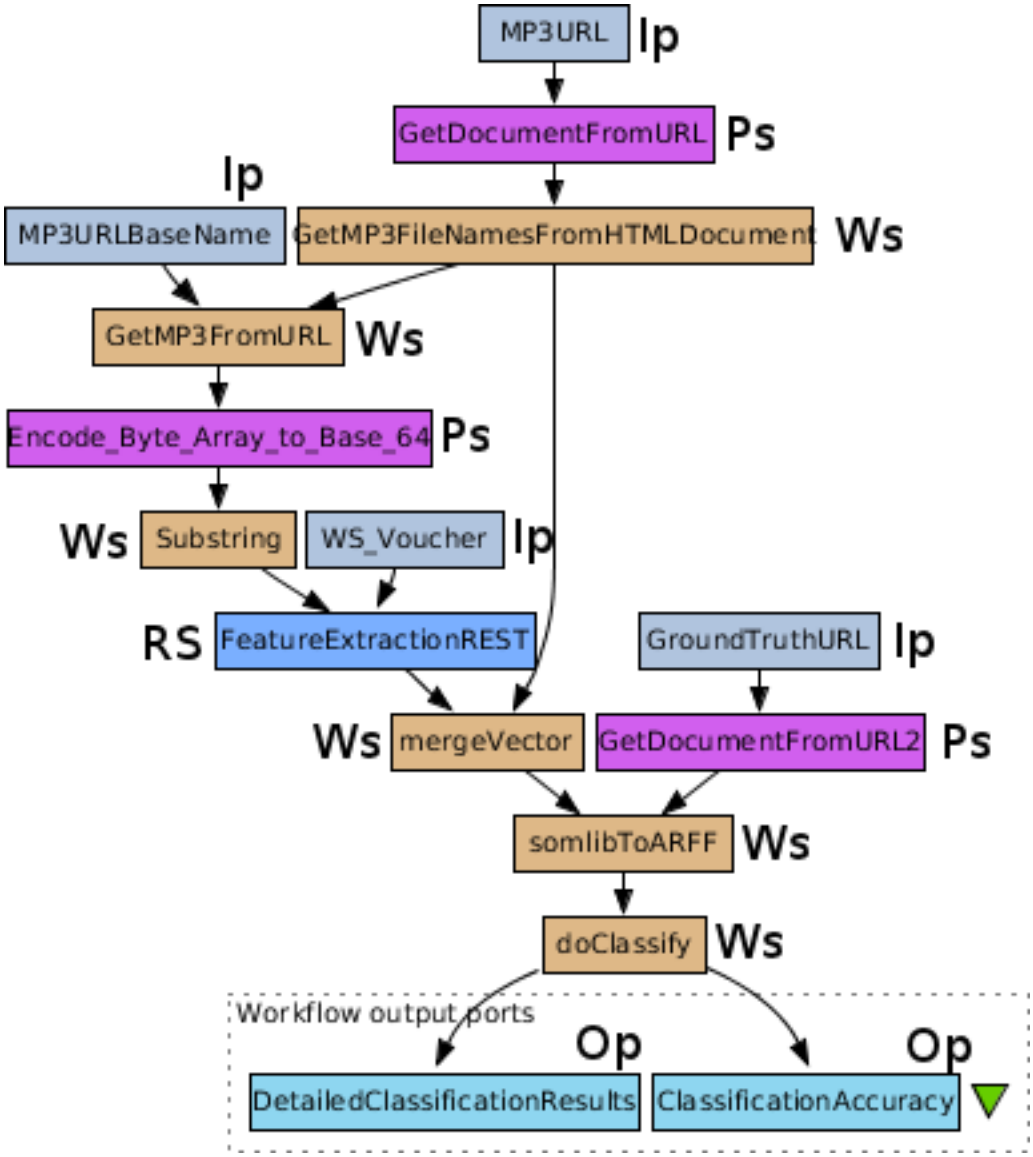

Figure 15: TIMBUS Music Process workflow in Taverna

Modelling the process in such an engine as it is based on Java has a side-effect of it becoming platform independent. First, steps that might normally be performed by shell scripts are replaced by a specific script-language known to Taverna. Also, all software components that are used have to be understood by the workflow engine, which thus becomes a layer of abstraction from the underlying operating system.

In detail, this process consists of and depends on the following software libraries or systems

- WEKA machine learning toolkit, version 3.6.6; employed for the learning of a predictive model and assigning of labels to unknown data
- Java SOMToolbox, version 0.7.5.1; used for format conversions
- Taverna Workflow Engine, version 2.3.0; used to execute beanshell scripts and provide the process workflow
  - Taverna requires Graphviz for rendering the workflow chain
- Java Development Kit / Java Runtime Environment version 6.0; use as runtime environment for Taverna Workflow Engineering
- Ubuntu Linux version 11.04; used as platform to run the JDK / JRE
- AudioFeatureExtraction REST Service, running at http://kronos.ifs.tuwien.ac.at:8080/fex/featureExtractionREST; provides the extraction of numerical features from MP3
  - CGI parameters:
    - voucher={authentication voucher}
    - music={mp3 file Base64 encoded}
  - Return value: Vector in SOMLib format
- MP3 Data provider Service; provides the audio files. For demonstration purposes, this is a simple Apache (version 2.2.0) directory listing, accessible at http://kronos.ifs.tuwien.ac.at/timbus/musicProcess/music/
- *Genre assignment (groundtruth) provider; provides the assignment of the audio files to a specific genre. For demonstration purposes, modelled as a simple HTTP service, available at* http://kronos.ifs.tuwien.ac.at/timbus/musicProcess/genres.txt*, in SOMLib format*

A screenshot of the dependencies graph of the TIMBUS Music Process that has been captured into the Formalism can be seen in Figure 16. This graph is produced from a Java graph visualisation tool called JUNG[48]. It gives more flexibility that the built in visualisation tool included with Protégé known as OntoGraf[49]. From this visualisation tool the relations that have been encoded between entities can be seen. Not all the relations have been encoded as this is the first version of the ontology and needs refinement based on the queries that will be developed as part of the subsequent deliverable. Some of the entities (including Ubuntu Oneric, ClassificationAccuracyGoal and MusicClassification) therefore are currently unconnected from the rest of the system and one of the aims of the iterative development of the Formalism is to

[48]http://jung.sourceforge.net/

[49]http://protegewiki.stanford.edu/wiki/OntoGraf

capture relations that describe the connections between all the elements that are necessary for digital preservation of a business process to succeed.

A graph of the packaging software dependencies that highlight the required packages in the system can be seen in Figure 17. These software package dependencies are captured using CUDF that was discussed in Section 3.3.2. The software package dependencies shown in the figure have been extracted from the system using the package management software 'APT[50]' and captured in CUDF. To visualise the dependencies the software 'debtree[51]' was then used. This shows the dependencies based on a Linux, Caixa Mágica 16 system that has a common basis of an Ubuntu 10.04 system. The limitations of software dependencies on their own is quite visible based on the tools that were used to capture this information. CUDF is limited is to describing software that is packaged, or contained in scripts. CUDF therefore cannot describe components such as software services, documentation, business process description to name just a few. To start with though, most software components on a GNU/Linux type system are captured as packages including system libraries and most commonly used applications. Most users will normally use software that is contained in packages but to aim for more complete coverage we have to be able to represent more generic software and also cover non GNU/Linux systems. In the scope of Workpackage WP6, TIMBUS will investigate and evaluate other tools for assisting users with extracting information and semi or fully automatically capturing instances and relationships as present in the live enterprise.

What can be seen is that some of the fundamental concepts of software service dependencies in Annex (Annex A.1 - Fundamental Concepts) from Linux software packaging systems have been modelled into the ontology and as such can be captured and represent information that is useful for determining if the model of the business process is complete enough for digital preservation. The CUDF representation is fairly limited in what it can describe and so the graph that can be seen in Figure 16 is able to represent a larger number of relations.

With the model that has been generated as joint work from Tasks T4.2 and T4.4, the TIMBUS Music Process has been converted into a representation that is consistent with the ontology. This captured version is being used to guide the tool development in WP6 and the work has also been carried out similarly for two other use-cases described in the scope of Task T4.4.

[50]http://packages.debian.org/squeeze/apt

[51]http://collab-maint.alioth.debian.org/debtree/

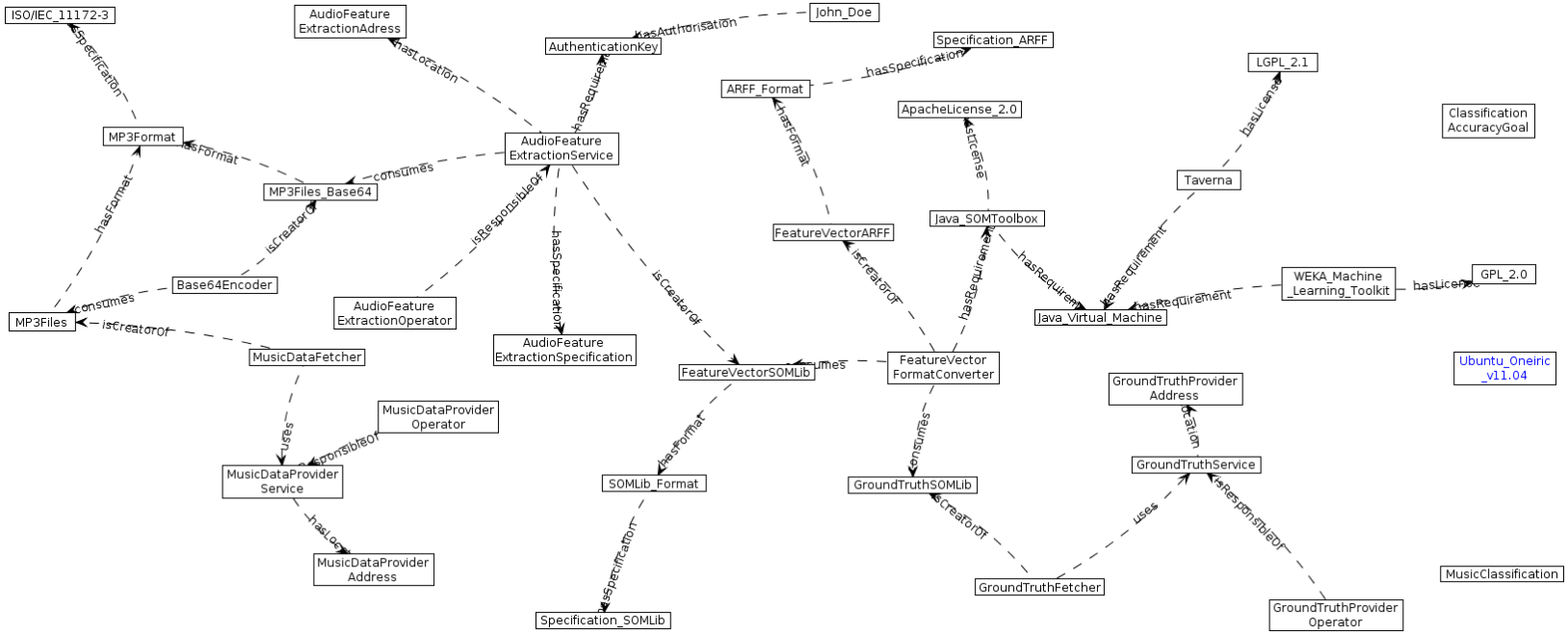

Figure 16: TIMBUS Music Process workflow as captured by JUNG

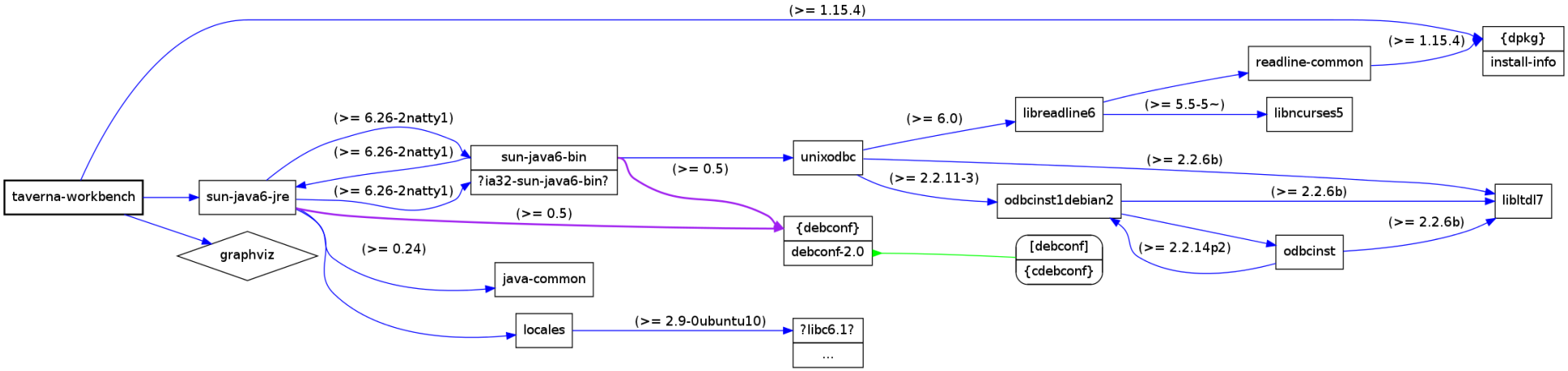

Figure 17: TIMBUS Music Process workflow software dependencies

6 Conclusion and Outlook

The TIMBUS project aims to enable the successful preservation of business processes, whose semantics are largely dependent on the context where the process was created and executed, requiring a particular environment in order to be understandable or rendered. This deliverable aims to contribute to that objective through the modelling of all the possible dependencies existing between digital objects that are part of a process or which the process depends upon, so that processes can be successfully exhumed and re-enacted.

In this deliverable a first version of an ontology for the modelling of the dependencies of business processes was proposed, which should be used in the remainder of the TIMBUS Project. The deliverable focused on investigating and reporting the related work in terms of descriptive terms that are used within different layers of an enterprise. This meant investigating areas that are not normally related and investigating possible representation formats for the context and dependency relations.

For representing the problem domain, pre-existing work was investigated for describing the relations and elements. However, the expressiveness of the surveyed specifications was deemed to be too limited or insufficient for describing elements in a business process for the scenarios that were investigated in conjunction with Deliverable D4.5. Moreover, extending the related work would sometimes involve creating new mechanisms which would create significant overhead, leading sometimes to almost complete reworking of the surveyed frameworks (e.g., extending CUDF would have led to creating new inference mechanisms and requiring that all the pre-existing solvers be modified to investigate the scenarios in TIMBUS).

As the scope of the scenarios is likely to broaden to include new relations and context parameters, a more expressive system for capturing and representing the elements of TIMBUS was suggested. It was suggested that developing an ontology would be a plausible way for answering the question posed in the problem statement, in Section 2.2, “What elements need to be captured in order to digitally preserve a required business process”.

It is assumed that the process for the generation of the context parameter and dependency models will be semi-automatic and as such it should be possible to capture the relations using natural language. Thus, the use of an ontology as the selected modelling technique should make it easier for model implementers to work with and therefore promote the usage of the Formalism. Modelling approaches such as TOGAF (Section 3.1.1) and Archimate (Section 3.2.1) to some extent capture similar

information but have a significant learning curve that make their use limited to skilled practitioners and require a large amount of overhead before any useful answers would be forthcoming. As such, the proposed approach which should be more intuitive and stripped down than full Enterprise Architecture methodologies was the proposed outcome of this deliverable.

OWL-RDF was investigated and adopted as the base-language for capturing the elements and their relations. Based on the scenarios investigated in the Context workshop and the expected relations required, using background knowledge from CUDF and DUDF, a first version of the ontology has been proposed. Two classes of dependency relations were derived from this work: constraint relations, which are strict relations that must be met in order in order to effectively preserve a process; and description relations, which associate entities with attribute details. The relations therefore were designed to be sufficiently expressive to capture the connections between entities whilst also being more intuitive for a human to model. From the investigation carried out in this deliverable it has been found that certain parameters and relations are explicit and thus easier to capture and store in the ontology, whilst other dependencies are not explicit and/or indirect and will be more complicated to represent.

This Formalism has been used to capture the TIMBUS Music Process and as such it has enough expressiveness to capture the properties of a technical system. This was the intention for the first deliverable that has a focus of looking at software dependencies and services.

6.1 Future work and D4.3 roadmap

Within Task 4.2 there is a second Deliverable, D4.3 that will allow for a set of refinements over the first iteration proposed in this Deliverable. In D4.3 the focus will be on taking the current version of the ontology, refining and extend it, and applying it to the use cases of WP7, WP8, and WP9, via tools being developed in Work Package WP 6. The limitations of the current model will also be assessed in the application of the ontology to the referred use cases.

A large part of the work will be on developing queries and rules for the reasoner such that it can work on a business process and based on the context parameters, determine the most relevant relations in the system, so that relevant context is captured and the process is successfully preserved. For certain types of business processes it is envisaged that certain contextual parameters will have more emphasis and as such the descriptive relations will be much more important for the reasoning.

Populating the Formalism with terms sufficient for representing the different aspects of an enterprise as identified in the related work in Section 3 will be an important step

for allowing the modelling of enterprises to be carried out semi-automatically. The mapping of elements in the various domain specific languages to the Formalism will allow for pre-existing tools that can extract specific information from enterprises to be entered into the overall TIMBUS architecture and populate the Formalism. Once in the Formalism, the various parts of the enterprise will have a representation that can then be reasoned upon and certain queries asked of the model. The model refinement that will broaden the scope of the context parameters and dependencies to encompass organisational and business components, as well as the level of detail of the components that can be incorporated into the problem set. Our approach for gathering dependencies is independent of emulation approaches however emulation environments provide a set of context parameters and dependency relations that will need to be captured. Most of the information presented by an emulated environment should resemble that of a real system and so the techniques that apply for real systems should be applicable to emulated systems but this will be confirmed in D4.3.

Handling hardware dependencies as well as some of the other concepts identified in the related work in Section 3, guided by being able to represent the scenarios identified in the context workshop of Task T4.4 as well as the use-case scenarios that are being developed in Work Packages WP7, 8 and 9 will also be one of the main focus points for Deliverable D4.3.

Additionally, the work that has been started in this deliverable will also be used as the basis for the practical implementations, D6.2 and D6.5.

Annex

Annex A.1 - Fundamental Concepts

In this appendix we will detail the parameters that are important for this task and define what is meant by them in this Deliverable. The definitions will also serve as a basis for other Deliverables and as such will be maintained on the TIMBUS glossary webpage[52].

Concept of Architecture

According to the [IEEE 1471:2000] Standard, architecture can be defined as “the fundamental organisation of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution”. In other words, an architecture has to depict the components of a system and their (internal and external) relationships, along with a way for governing their evolution. In this sense, it can be observed that architecture could be used as a tool for capturing components of a system and their dependency relationships with each other and with the business environment. Since the task of capturing all the components of a system can be cumbersome, the idea of using viewpoints as a means of dividing a big problem into smaller ones that are less complex to solve is a common practice in architecture development. A viewpoint is itself a magnification of a part of tan architecture from the perspective of a stakeholder of a system, resulting in a subset of the system components and relationships that answer his concerns.

Enterprise Architecture

Enterprise Architecture can be seen as a way of “modelling the role of information systems and technology on the organisation, aligning the enterprise-wide concepts, aligning the business processes and information with the information systems, planning for change, and providing self-awareness to the organisation“ [Sousa P., et al., 2006]. The concept of Enterprise Architecture is similar to that of Architecture, but with a larger focus, which encompasses organisations and their information systems, and consequently has to address a larger number of internal and external stakeholders. In that sense, it aims to present a holistic view of the organisation, of the components that compose it, and their (internal and external) relationships, which is not confined to business, but also encloses technology and systems.

[52]http://timbus.teco.edu/projects/glossary/wiki – TIMBUS Glossary page

Business Process Modelling

Business processes are the core of the information to preserve and the glue between the information artefacts. A business process is an ordered set of activities needed to fulfil a business goal. It also includes artefacts as input and output of activities and roles for doing activities [Aguilar-Savén R., 2004].

A process can be defined as the "relationships between inputs and outputs, where inputs are transformed into outputs using a series of activities, which add value to the inputs" [Sousa P., et al., 2006]. On the other hand, a business process has a larger scope than that of a process, since it is an ordered set of activities needed to fulfill a business goal, including the input and output artefacts of activities and roles for doing activities. In that sense, the importance of modelling business processes comes from the fact that the more well known is a business process, more manageable and improvable it becomes. Several techniques for the modelling of business processes exist, such as flow charts, IDEF (Integration Definition), UML sequence/activity diagrams, BPMN, among others.

The Concept of Service

A service at the base level is a set of operations performed by an entity at the request of another entity that takes a set of inputs, is influenced by a set of environmental conditions (e.g. SLAs, for example on timing), also called contexts of the service, and produces an output in a format that is well defined. For a service to function correctly it is important that a service has a well established interface between the entities. Depending on the context of service the complete functional specification of the service may be known as the protocol, contract or through other terms.

Generic Services

There are many types of service that are applicable to an Enterprise Architecture. Each depends on the domain to which they are being applied. For this Deliverable the focus will be on services in relation to software. The second deliverable associated with this task will look at relating business services with software services. Using a relation between business services and the underlying technology layer in terms of software, we will investigate linking business and software services.

Services in Business/Enterprises

A Business service can be defined as a subset of a service that takes into consideration business requirements. A Business service is a set of business operations performed by a business/enterprise at the request of another through a formal interface. The main difference between a software and a business service is that a business service interface tends to be controlled and governed by an

organisation, whereas for a software service the interface is normally common and standardised outside of the domain of either the client or the server.

In The Open Group Architecture Framework (TOGAF) Business and IT services are distinguished;

- Business service: Supports business capabilities through an explicitly defined interface and is explicitly governed by an organisation.
- Information System Service: The automated elements of a business service. An information system service may deliver or support part or all of one or more business services.

Services in Software Engineering

In Software Engineering a service can be started by a user, process or application and has certain access conditionality associated. Once activated, other clients can use access credentials use the software service to perform an operation. Depending on the set of other users running and how the service is designed, an output will be produced and returned via a known set of output channel(s) to the client.

An example: A web-server, Apache that is run by user apache on a Linux System can then be accessed via other users locally or remotely depending on the configuration, normally through port 80 to view HTML pages through the established protocol of HTTP (Hyper-text transfer protocol). The input to the service is a URL (Uniform Resource Locator) request that the service will then process and produce an output compliant with HTTP, allowing the user to see a webpage.

Dependencies

Dependencies in general are a descriptive term for relationships between two entities. If entities are represented as nodes in a graph, then dependencies are the edges between these nodes. The type of relationship as denoted by the edge is determined by what the graph is representing though. There may be many similar types of relationships that can be linked in this way. In general:

Entity_1	Relationship	Entity_2

Where Entity1 and Entity2 are two different entities or different specific instantiations of entities and Relationship denotes the type of relationship the entities have on each other. The order is important because the relationships can be directed.

The definition of a dependency is actually more specific than the general entity relationships and it denotes a requirement that one entity must be present at the same time as the other. An entity that is dependent on another is known as the dependent and the entity to which it refers can be called the dependee.

The list of relationships in Table 6, demonstrates a series of types of relationships that may exist between entities. These are derived from the work carried out in the development of Linux Software dependencies but when abstracted can be applied to most other relationship types between entities.

Table 6: Relationships between entities

Name of relationship	Example	Meaning
depends/requires	Entity1 depends Entity2	Entity1 can not exist in the system without Entity2 existing at the same time.
conflicts	Entity1 conflicts Entity2	Entity1 or Entity2 can exist on the system but cannot exist at the same time. Entity1 and Entity2 are mutually exclusive.
recommends/suggests	Entity1 recommends(weighting) Entity2	Entity1 does not require Entity2 to be in system, to exist but provides a suggestion as to how important it might be, depending on the value of weighting.
pre-depends	Entity1 pre-depends(time) Entity2	Entity1 requires that Entity2 be in the system at least for time before it can exist in the same system. Entails an ordering of dependencies.

At a business level, the management of the dependencies of a business process is a critical aspect in the business/IT alignment effort. Processes contain implicit and explicit intra- and inter-relationships with other components which in fact are recognised and modelled by the majority of business modelling languages. Despite not being labelled as such, it can be observed that several relationships between those components are in fact dependencies, e.g., the realisation of an activity in a business process by a software component (i.e., cross-layer dependency). On the other hand, these general purpose modelling languages do not classify dependencies, which brings problems when trying to enforce traceability in the architecture.

Safoora et al. [Safoora, K., et al., 2008] partly resolve this problem by defining seven types of dependencies between requirements and the architecture:

- Goal Dependency: Relates the system quality attributes at problem domain to their realisation in the architecture and implementation;
- Service Dependency: Relates requirements and the operations and functions of the architecture;
- Conditional Dependency: Relates conditions, constraints, and decisions taken at the requirements levels to the architecture;
- Temporal Dependency: Relates requirements specifying the time frame of an event to occur, processes to complete, etc., to their realisation in the architecture;
- Task Dependency: Relates requirements specifying the connection between tasks (response, input, feedback, etc.);
- Infrastructure Dependency: Relates the technical/realisation requirements to their specific realisation in the architecture (resources, infrastructure, standards, etc.)
- Usability Dependency: Relates requirements concerning user interaction with the realisation at the architecture level and implementation.

Software Dependencies

The definition of dependencies in software engineering has two meanings. One refers to the taxonomy of a set of relations between components. A second definition is when a component is required by another and as such is dependent on that other component.

The generic term 'dependencies' is used to describe relationships between components because normally it is the most common relationship held between entities.

Normally when dependencies are discussed in this document in general they include the set of relations but if the word is used in relation between two or more components then it will mean that one of the software components is required by the other.

Package relationships or dependencies as defined above can include the following relations when applied to software packages:

- depends - pkg A depends pkg B: if pkg A to be installed, pkg B must be installed on system.

- recommends - pkg A requires pkg B: if pkg A to be installed, pkg B should be installed on system.
- suggests - pkg A suggests pkg B: if pkg A to be installed, pkg B could be installed. Weaker than recommends.
- enhances - pkg B enhances pkg A: pkg B could be installed with pkg A. Weak preference.
- conflicts - pkg A conflicts pkg B: if pkg A to be installed, pkg B cannot exist on system.
- provides - pkg B provides “service” and pkg A depends on “service”. If pkg A is to be installed then pkg B or some other package must be installed to provide “service”.

Source: Debian Dependencies[53]

Software Dependencies are a way of using the concept of reusable components on a system. Components can call other components and are designed to be called in a well established manner. By having an API (Application Programming Interface), one piece of software can interact and use the result of another. Component based systems such as Linux rely on this for allowing software to be generated by multiple developers across the world to use standard features required by software to be used by multiple software applications, without having to redevelop all the original work. A dependency is an encoding that can be added to software, most normally in Linux through Packaging meta-data, which allow for applications to rely on functions and definitions that are defined in other pieces of software, without having to directly include them. By using package dependencies, the system can quickly check to see if software will be supported given a configuration of software on a system. This is handled by a package manager that captures the relations between software packages.

For the purpose of describing software dependencies an Operating System created by the developer Microsoft and generally named in the format Microsoft Windows X (where X may be non-exhaustively, 95, 98, ME, SE, 2000, XP, Vista, 7) will be shortened to Windows. Also given that most users in an Enterprise or Consumer environment use more modern versions of Windows, it will be assumed unless stated otherwise that Windows refers to one of the operating systems based on New Technology (NT). If the differentiation of a particular Operating System type is required then 'Server' or similar will be appended and if a part requires a particular version (Home/Ultimate/Corporate etc.) then this will also be stated.

[53]http://www.debian.org/doc/debian-policy/ch-relationships.html

For describing the software dependencies of an Operating System created by the developer Apple and generally have the name System 1-7 or Mac OS 8-X, will be shortened to Mac OS. Most usage of Mac OS revolves around the use of Mac OS X. If a particular version such as 'System 6.0.8' or 'Tiger' is required then it will be stated explicitly.

Hardware Dependencies

The computer industry strives to minimise hardware dependencies through the adoption of common architectures and standards. In the earliest days of computing, moving a software package from one platform to another would certainly require code modifications before being recompiled and tested. The development and adoption of, for example, the x86 processor architecture, has provided a platform upon which software vendors can build reliable, predictable and affordable operating system environments. The wide range of capabilities offered by these systems has arguably been a strong driver of the success of the personal PC and all that has come after that.

Today, the effect of hardware dependencies is reduced, but it has not disappeared. For the casual IT consumer, it may be barely noticeable as they migrate between commonly available software packages on current generation operating system platforms without paying much attention the dependencies that those packages may have on the underlying hardware. For software developers, especially anyone with heavy compute needs, graphic processing, or niche requirements, hardware capabilities can make a significant difference to the performance of the application and hence the end-user experience. Over time though, every IT user becomes aware of the effects of hardware dependencies as it becomes impossible to execute previous generation operating systems and their software stacks on current hardware. This effect is lessened for virtualised computer systems, but it still does not go away as even a virtual machine presents a set of emulated hardware to the operating system. A possible solution to this is to increase the array of legacy emulated hardware available within hypervisors over time.

In the context of digital preservation, and specifically in the approach of the TIMBUS project, the identification of hardware dependencies is a crucial component in understanding some of the major inhibitors to exhuming archived execution environments on emulated or virtual infrastructures. Even when considerable care has been taken to address these in the past in recent examples such as the Digital

Domesday Book[54], within a very short timeframe, the hardware required to read and interpret the data has become obsolete. The problem is two-fold:

1. Media access:

Arguably the modern IT industry has its roots in the late 1960's when the foundation for the widespread adoption of personal computers was laid. Within that time, lets consider some of the types of media and transport/storage technologies that have been created. How many of these can still be commonly accessed today by readily available computer technologies: Tapes, floppy disks, CDs, DVDs, USB/Firewire/Lightpeak, SCSI/SAS/SATA/SSD, ethernet/FDDI, IPv4/v6, PCMCIA, SRAM/DRAM, the list goes on. How many do we think will be accessible in 10 years time. Certainly within 20 years time all these can be expected to be superseded. The first issue with hardware dependencies is literally the problem of the technologies upon which we rely for the storage and transportation of data.

Over time media will deteriorate even in a proper climate controlled environment. Intel has seen cases where just the breeze caused by walking past backup tapes caused some of the plastic hooks on reel to reel tapes to break which brought the tapes crashing to the floor. It is therefore clear that not only does the actual tape deteriorate, but the container in which the tape is housed becomes brittle over time. The same can be said of other forms of media. Tapes from the late 1980s were not as brittle as those from the 1970s, so it would imply that the plastic housing of existing media has a roughly 30 year lifespan and it should be remembered that is based on the assumption of correct climate controlled storage.

2. System Architectures:

Every operating system is developed to run on specific sets of hardware at a certain point in time. Most operating systems today specify a hardware compatibility list (HCL). The HCL simply lists the hardware components for which drivers have been developed allowing that operating system to run on that hardware. Operating system versions advance in lock-step with hardware advancements creating an unseen, and very short time-window within which these layers can be expected to reliably operate. This issue also effects software running within the operating system which may have dependencies on specific hardware features and capabilities. This not only applies to classic server and PC systems, but it applies equally to new form factors such as smart phones and tablets as well network switches and sensors which are all important parts of todays business processes. If the past has shown us anything it is that evolution of hardware is accelerating and there is no reason to think that the business processes of the future will not include an even wider array of future devices.

[54]http://www.atsf.co.uk/dottext/domesday.html

To ensure the best chance of success against both of these issues, hardware dependencies must be understood and addressed at the time that the archival information objects are created. This is the best opportunity in the entire archival process to allow for future exhumation by understanding these dependencies and making allowances for their support at a time when the ability to do something about it is still possible. The development and adoption of virtualisation technologies has provided a layer which can abstract running software from the specific underlying hardware through the provision of a set of emulated hardware components provided by the hypervisor.

The dependency analysis aspect of the TIMBUS project will therefore also need to examine how hardware dependencies will affect the archival and exhumation process. The broad areas that need to be considered for an emulation based digital preservation solution include:

- The intersection of hardware compatibility lists (HCLs) for all current and legacy operating systems in our environment with the offered set of emulated hardware available in current generation hypervisors. If these do not intersect or overlap, we have a gap in our ability to exhume an archived environment.
- Specific applications may have specific hardware requirements. Ultimately these can all be categorised as performance enhancers. Examples of these may include:
    - any application that takes advantage of an x86 extension (eg: MMX, AES-NI, VTx, etc) to improve performance
    - high end graphics generation or display may require extra video processing capabilities to be supported by hardware
    - some applications can require specific bandwidth is available to its network or disk operations for optimal operation.
- Media and transport-layer dependencies.
- Components outside the classic client/server HW set such as smart phones, tablets and sensors. These not only consume data, but can generate and manipulate it as well.

Hardware dependencies are a complicated aspect of an enterprise but however are not the particular focus of this Deliverable. With the re-emergence of virtualisation and the usefulness of emulation already being discussed in the digital preservation community it will be one of the points to focus on the subsequent Deliverable, D4.3,

once some effort has been carried out into the Deliverable that focuses on how virtualisation will be handled from an architectural perspective in D5.3 (Architecture and infrastructure definition for virtualisation, storage, rerun and integration (VSRI)).

Configuration Management

Configuration management controls all aspects of a system or product in relation to its requirements, design and actual performance throughout its lifecycle. Thus, the term “configuration” is used here as the combination of all parts of a system rather than only the parameters of a software, including software and hardware (however, not necessarily making their dependencies explicit).

The most relevant form of CM appears to be software CM (SCM), where some widely disseminated definitions are illustrative: that from the Institute of Electrical and Electronics Engineers (IEEE) and the Software Engineering Institute (SEI) at Carnegie Mellon University (CMU).

The IEEE standard standard 729-1983 [updated as IEEE Std 610.12-1990] is illustrative of the scope covered by CM:

- *Identification*: “identifying the structure of the product, its components and their type, and making them unique and accessible in some form”
- *Control*: “controlling the release of product and changes to it throughout the lifecycle …”
- *Status Accounting*: “recording and reporting the status of components and change requests, and gathering vital statistics about components in the product "
- *Audit and review*: “validating the completeness of a product and maintaining consistency among the components …”

This is complemented by a definition as part of CMU’s Capability Maturity Model (CMM) - "…Software Configuration Management involves identifying the configuration of the software (selected software work products and their descriptions [= *identification*]) at given points in time, systematically controlling changes to the configuration [= *control*], and maintaining the integrity and traceability of the configuration throughout the software lifecycle. The work products placed under software configuration management include the software products that are delivered to the customer (for example the software requirements document and the code) and the items that are identified with or required to create these software products (for example the compiler)…", [CMU, 1995].

Furthermore, the definition within the widely used Rational Unified Process (RUP) includes: "*The task of defining and maintaining configurations and versions of*

*artifact*s. This includes baselining, version control, status control, and storage control of the artifacts.", [Dumbill, E., 2012]. For additional definitions, cf. [Norin J, 2007], [Bamford, R., et al., 1995], [Babich, W., 1986], as well as the definition of the International Organisation for Standardisation (ISO).

From a management perspective, the principles and practices of CM represent an accepted and understood foundation for implementing ISO-compliant processes in software engineering organisations.

Because CM complements the views on software and hardware dependencies described above not only by contributing the missing parametrisation of IT systems but by viewing them more holistically, in the optimal case the practices, tools and methodologies used in CM may yield a valuable basis to develop DP systems.

Context

A definition of context that has been accepted widely in the area of context-aware applications has been given by Dey and Abowd: “Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.” A problem with this definition is that it does not differentiate between context and information about context. A more recent definition by Bardram better reflects the fact that context exists outside a representation system: “ ‘Context’ refers to the physical and social situation in which computational devices are embedded.”

A context model can then be conceived of as a model suitable for storing information about the context of a certain interaction event. Mobile context-aware computing has to cover issues of sensor reliability, ad-hoc network communication, software development support, reasoning and inference, usability, and privacy management. Most recently, ontology-based approaches have gained importance to answer the demands of these heterogeneous application environments with regards to interoperability of context models. The key idea of ontology-based context modelling is that applications using the context model also have to agree on a common ontology, that is, a set of basic concepts defined in a formal language, which developers can use to specify application specific concepts. Application concepts, being founded upon the same basic concepts, can then be used for communication between different applications.

When human beings reason or communicate about entities in the environment, they usually abstract from certain aspects, and reason or communicate within a certain

context. Therefore, their reasoning and communication depends on the context the entities are in. Without the dependant context, their reasoning or communication, in worst case, may look absolutely unreasonable or “out-of-context”, so to speak. When we reason about space, for instance, we may use *to the West of* as a transitive relation. This assumption is valid as long as we suppose a sufficiently small local area of context, as *to the West of* is globally a cyclic relation: Denmark is *to the West of* Korea, Korea *is to the West of* Canada, and *Canada is to the West of* Denmark; within the local context of a city or country, in contrast, *to the West of* can be used in the same manner as *to the North of*, i.e. as if it was a transitive, acyclic relation.

Context Dependencies

As a starting point, we assume a general concept of context (which is composed of context parameters) as the situation that an entity (in focus) resides in. In accordance, context information refers to any information describing the situation (meaning, information that describe the context and the context parameters it is composed of) of an entity in focus. A (somehow) observed fact that an entity depends upon its context is called context dependence. Furthermore, context parameters (or context aspects) can depend on each other. For example, the time zone in which an event is performed depends on the event’s geographic location. Another example is the technologic platform (hardware and software) a business process depends on to be successfully executed.

Domain Specific Languages

The following section is adapted from [Voelter, M., 2009].

A domain specific language (DSL) is a language that has been defined for a specific domain. In other words, the concepts used by those languages are specific of the domain they are covering. A domain specific language can be used by developers and architects, but also by business users, and can be used as input for code generation, validation, simulation or interpretation. When building a DSL, either a technical or a business one, the knowledge of the domain already exists, tacitly or implicitly, only needing to be captured and formalised. It should have a limited expressiveness.

As oppose to General Purpose Languages (GPLs), DSLs have to deal with a reduced number of concerns [Fritzsche, 2010]. Examples of GPLs are General Purpose Programming Languages like Java and C and the General Purpose Modeling Language UML. On the other side, concrete examples of DSLs are the modeling language Modelica [Mattsson, S., et al., 1997], as well as the Business Process Modeling Notation (BPMN).

A DSL consists of abstract syntax, concrete syntax, and semantics. The abstract syntax is the result of the formalisation of the concepts of the domain being analyzed,

resulting in a metamodel. The concrete syntax should be adequate to the domain of its users, so that its use and adoption is free of trouble. It can be either textual or graphical, depending on the domain being analysed. For instance, graphical notation is adequate for demonstrating relationships between entities, data flows, etc. (e.g., UML can be considered concrete syntax). The language semantics takes into account the knowledge from the domain. Semantics provide meaning to the language, and are often described in prose and usage examples and embedded on the platform which will compile/interpret the language.

Also important in the definition of a DSL is the concept of viewpoint. If the purpose of a language is to describe a whole system, then it should make use of viewpoints describing different concerns of a system, providing notations and abstractions for each. Viewpoints should also be related with each other so that the description is coherent.

The benefits of using DSLs have been quantitatively evaluated in [Kieburtz, R. et al., 1996], in which a higher degree of efficiency, e.g. in terms of productivity, was revealed. It turned out, DSLs are beneficial if a family of programs, such as the family of business processes, are addressed, which might occur again in the future. Also, it is claimed that with the help of DSLs the constructs are usually more high level and are therefore typically significantly shorter than their pendant construct in GPLs [Jouault, F., et al., 2006]. For example, assuming a process model is built using a business process DSL, the individual activities could be automatically generated as service, given that some further information is provided. As a result, developers using a DSL can concentrate on creative tasks rather than repetitive tasks. Another benefit of using DSLs is pointed out in [Fritzsche, 2010] and displayed in Figure 18. In order to involve a domain expert who is not necessarily a good skilled programmer, it is proposed to split up the responsibilities of development into three different roles:

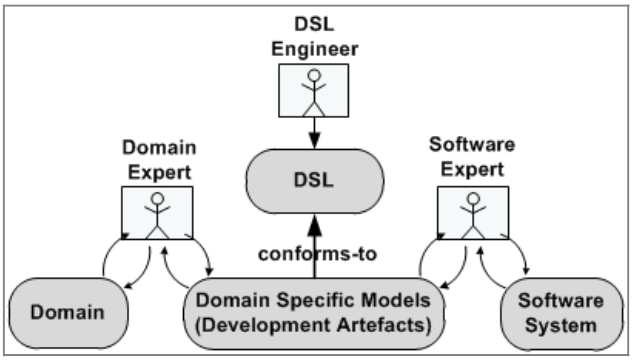

Figure 18: Expert roles using Domain Specific Languages

- The DSL Engineer defines the DSL, e.g. BPMN, and needs to provide tools to create instances of this DSL, e.g. BPMN Modeling Tools.
- The Domain Expert uses the provided tools to define and manage his instances of the DSL. Examples of domain experts are business process analysts or business impact analysts.
- The Software Expert is the one who actually implements the development artifacts. In case of an activity element in a business process model, he would be the one developing the service using a GPL, such as Java. Preferably, this implementation process is automated in the best possible way.

With the proposed sharing of responsibilities, domain specific models can be used as a mean of communication between domain expert and software expert. Now, the domain expert is able to focus on his respective concerns which is, to conclude, one way to deal with the complexity of modern systems, e.g. process environments.

Information Collection

Information as defined by [ISO/IEC 2382-1:1993]:0.1.0.1.01: Knowledge concerning objects, such as facts, events, things, processes, or ideas, including concepts, that within a certain context has a particular meaning.

Data as defined by [ISO/IEC 2382-1:1993]:01.01.02: A reinterpretable representation of information in a formalises manner suitable for communication, interpretation, or processing.

Data collection as defined by [ISO/IEC 2382-6:1987]:05.02.08: The process of bringing data together from one or more points for use in a computer.

Data entry as defined by [ISO/IEC 2382-6:1987]:05.02.09: The process of putting data onto a machine-readable medium.

Data acquisition as defined by [ISO/IEC 2382-6:1987]:05.02.10: The process of collecting and entering data.

Data collection has migrated from the 1960s view through, Data Access 1980s, through Data Warehousing and Decision Support 1990s up to Data Mining 2000s, source [Coxon A., 1999] [Weimer, J. (ed.), 1995] [Weller, S., et al., 1988] [Sapsford R., et al., 2006].

Information collection requires a few stages that should be defined as well.

Information collection can use data gathered through data acquisition for sampling information from sensors to the physical world. It can also use information from data extraction systems that process a set of input sources in a non or semi-structured system. This requires the identification of the sources for collection. This may require a preliminary information collection for pre-processing. Scoping includes identification of data-sources and identification of how data elements are structured. ETL is the process of extracting the original data sources, transforming them as required and then loading them into a representative format that is useful for the rest of the system. Transforming is defined as changing the form of data according to specified rules, without fundamentally changing the meaning of the data ([ISO/IEC 2382-6:1987]:06.03.04). To load is defined as transfer [of] data into storage device or working registers ([ISO/IEC 2382-6:1987]:06.03.03) and transfer is defined as to send data from one storage location to another ([ISO/IEC 2382-6:1987]:06.03.01).

A methodology that is often used is that of “Extraction, Transform and Load” (ETL) of information. Information collection will be carried out throughout the dependencies and context extraction and mapping that will be performed in TIMBUS. Decisions of the level of abstraction to use and the level of dependencies all relate to the original components that are identified as discussed in the methodology in Section 2.4.

Another definition used for the acquisition of information from media sources is that of “Automatic Identification and Data Capture” (AIDC). AIDC is used for obtaining external data normally through analysis of images, sounds or video.

A boundary of digital preservation and therefore of the TIMBUS process itself is in how to manage the scope of information and how to deal with unstructured data. For the purposes of defining unstructured data we will use the definition that unstructured data is “Data that cannot be easily represented as a model or if a model exists the set of data that belongs to the class of model is small”. Easily and small are two terms that weaken this definition but as can be found in certain articles there are approaches to apply structure to seemingly unstructured data, [Buneman, P., et al., 1997].

An email header for example can be seen as structured data as can the payload of the data contained within the email. However the text in the email as it is normally written by a human will normally be unstructured. The Internet is often also cited as an unstructured data-source but several layers of structuring can be applied that would make a more concrete definition difficult to maintain and use.

Information stored in the data of the system to be preserved is the basis for many of the methods and techniques that will be used throughout TIMBUS. The importance of collecting data for the manipulation of information for use in preservation can be

summarised in a quotation from Sir Arthur Conan Doyle as Sherlock Holmes, “It is a capital mistake to theorise before one has data”.

Annex A.2 - Dependencies in Service Operation and Lifecycle Processes

In order to be able to model and capture dependencies there are several IT industry standards and frameworks regarding or making use of configuration management. Configuration Management [ANSI/EIA-649-2004] is a standard function in different contexts such as ISO 20000 [ISO/IEC 20000-1, 2011] (IT service management, ITIL), IEEE 828 (software configuration management plans), ISO 10007 as a supplement to ISO 9001 (Guidelines for configuration management). Further ISO/IEC/IEEE 12207 and 15288 (Software Lifecycle Management) and software engineering best practises typically produce a number of artefacts on dependencies that are also valuable for TIMBUS purposes. These standards may not apply to all the company needs and so they might require tailoring according to the company's needs.

For TIMBUS, configuration management data can be a valuable resource of dependencies between various types of artefacts. It is therefore useful to take into account the configuration information typically available in organisations complying for following the above mentioned standards and leverage the information for digital preservation purposes. Further sources of information are produced during the software lifecycle.

Details of the respective standards are described briefly in the following sections.

Configuration Management in ITIL and ISO 9001

The Information Technology Infrastructure Library (ITIL) is a vendor independent, set of best-practices for IT service management, owned by the United Kingdom's Office of Government Commerce (OGC) [OGC-ITIL, 2007]. It describes a set of processes to manage the IT services and the underlying IT infrastructure in an efficient and effective way with the aim of fulfilling service level agreements made with internal or external customers. Configuration management is a key function for ITIL and information necessary for operating services are stored and managed in a Configuration Management Database (CMDB). The CMDB has the aim of tracking all components of the IT infrastructure, including software, hardware, and documentation, configurations, and the relations existing between these items [CMDB, 2010]. The CMDB plays a major role for further processes in IT service management such as Change Management, Disaster Recovery etc.

Configuration Management in ISO 9001

ISO 10007:2003 [ISO: 10007:2003] is part of the ISO 9001 [ISO: 9001:2008], designed to support a quality management system which provides guidelines on the use of configuration management by defining the configuration management process. This

includes configuration management planning, configuration identification, configuration status accounting, and configuration audit. Configuration management includes baseline identification and version control, status reporting, and change control as well as detailed roles and responsibilities.

Software Configuration Management Plans (IEEE 828-2005)

The IEEE 828-2005 [IEEE: 828-2005] provides the most widely used guidelines for creating a software configuration management plan. The activities defined in the standard include configuration identification, configuration status accounting, configuration control, configuration audits and reviews, subcontractor and vendor control, as well as release management and delivery. Further requirements include the necessity of having processes for creating baselines and change control for all configuration items.

Software Lifecycle Management

The ISO/IEC 12207 [ISO/IEC 12207, 2008] details the software engineering processes used in the lifecycle of a software system. ISO/IEC/IEEE 15288 [ISO/IEC 15288, 2008] details all the processes used in the lifecycle of a human-made system. These standards provide a framework for the development phase of software products, and contain a “treatment” for any fragment of the development lifecycle. Also, they can be extended by using any of the quality assurance, code reviews, and configuration management planning and testing ISO/IEEE standards. Both standards are an typical starting point for the other specific standards detailed above.

There are further frameworks offering guidance on establishing CM-related practices, including COBIT, CMM/CMMI (Capability Maturity Model Integration), and the Software Engineering Body of Knowledge (SWEBOK). In SWEBOK, software configuration management is defined as the discipline of identifying the configuration of software at distinct points in time. SWEBOK is publicly available [IEEE-SWEBOK, 2004] and is synched with all the IEEE standards, periodically.

Annex A.3 – Dependency relations as mapped by IBM Rational Software Architect

IBM Rational Software Architect (IBM RSA) is a comprehensive modelling and development environment that uses UML for designing architecture for C++, J2EE applications and web-services. It is built on top of Eclipse Software Framework and includes capabilities focused on architectural code analysis, C++ and Model Driven Development[55]. The version discussed here is 7.5, which supports UML 2.1 and model based development. Of interest are the dependency modelling and modelling capabilities. It is based on a fairly simple core model which is depicted in Figure 19.

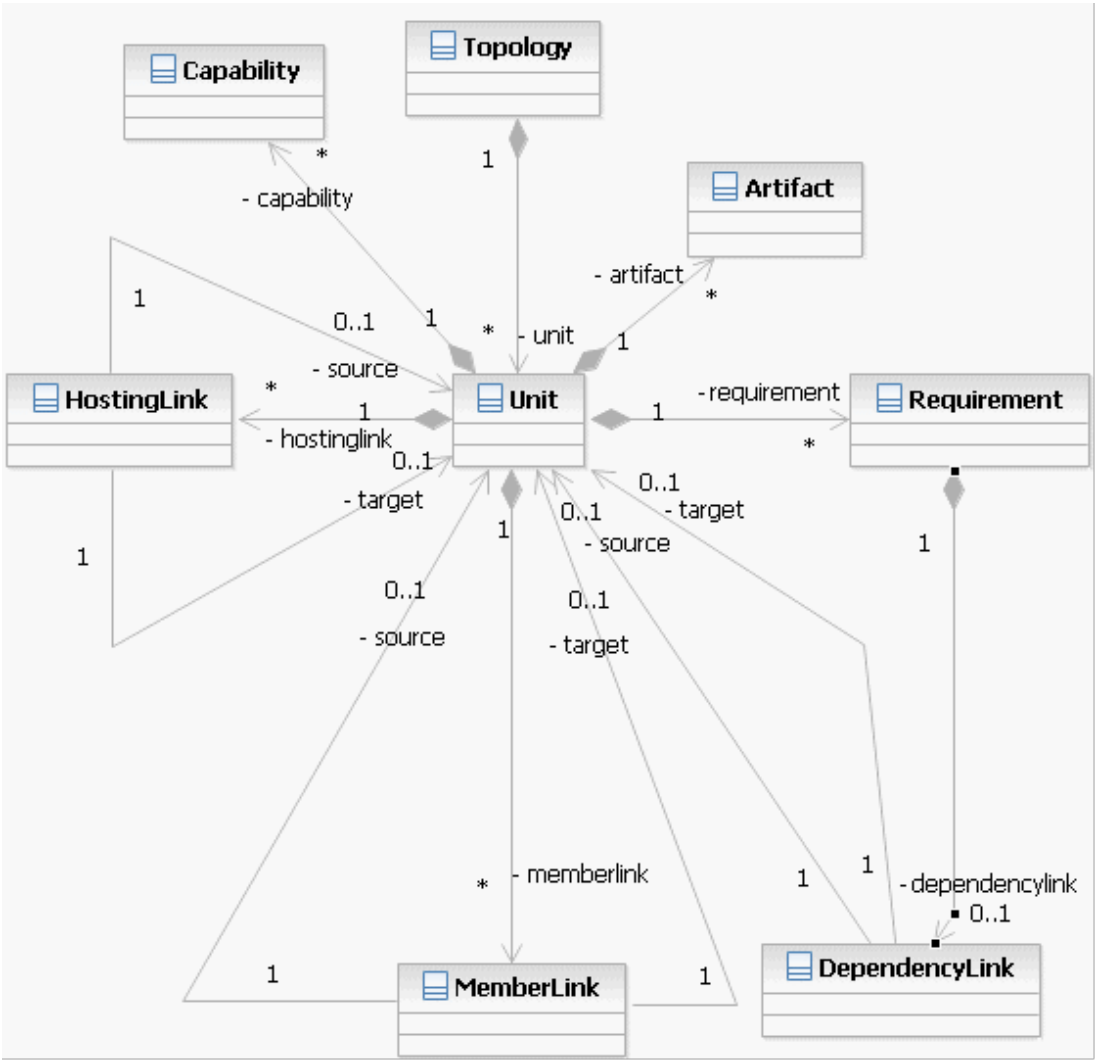

Figure 19: Basic topology UML model of IBM RSA[56]

[55]http://ibm-rational-software-architect.software.informer.com/wiki/

It can be used to model an IT system or parts of an IT system. Despite the core model's simplicity, it is designed in a way that new types, with respect to a domain of the topology that is to be modelled, can be easily injected. Such extensions exist for a couple of domains, e.g. server, storage, OS, database, and J2EE.

Elements of the core model are:

- Topology is the top-level container of the topology.
- Unit represents the unit of deployment, i.e. it is your basic model entity (more specific units are defined in the model extensions, e.g. File System, Operating System).
- Capability is to describe an ability that is offered of a unit.
- Requirement is to describe a need that is required by a unit. It will be checked against offered capabilities of the connected units.
- Artefact describes a deployable resource or an object.
- Relationship Links are the connections between the unit. There are a couple of different links, three of them in the basic topology model shown in Figure 19. As they are of relevance for the dependency relations they are discussed in slightly more detail in the next subsection.

Dependency relations taken into consideration

Relations in the topology model are defined through Links. Links are specified at an abstract level then more fully specified. Relationship links are important for the dependency model as most of them resemble one particular kind of dependency in the dependency model. Three types are defined at high level: Dependency, Hosting and Member Link. Each of these links hold the source and the target of the link that is participating in that relationship.

A *Dependency Link* is used to link a requirement with a link of dependency to a capability, which indicates that the requirement is fulfilled by the target capability. It also provides a means to enforce compliance of the source requirement against the target capability.

A *Hosting Link* indicates that a unit will be hosted based on the fulfilment of all hosting requirements with hosting capabilities on the target host unit.

[56]Makin, N. Anatomy of a topology model in Rational Software Architect Version 7.5: Part 1: Deployment modelling.
http://www.ibm.com/developerworks/rational/library/08/1202_makin/index.html
http://en.wikipedia.org/wiki/System_Architect_(software)

A *Member Link* represents the containment relationship that defines the linkage between two units, where the target of the link is the member and the source is the container.

There are more complicated relationship links of which some are briefly described in the following list:

- A *Realisation Link* is used to link a conceptual unit with another unit to indicate that the target unit will replace ("realise") the conceptual unit. As this is basically the class-instance-relationship from the object-oriented world this link does not translate into a dependency. Conceptual units simply do not exist in a implemented software systems.
- A *Constraint Link* is used to constrain the source and the target unit based on the semantics of the child constraint placed on it. As opposed to the realisation link a constraint link is of value for a dependency analysis. It can be mapped to similar concepts in the configuration management.

Including the extensions available, which are defining more specific units and relationship links, the components and inter-relationships of a software system can be modelled quite thoroughly.

Implementation example

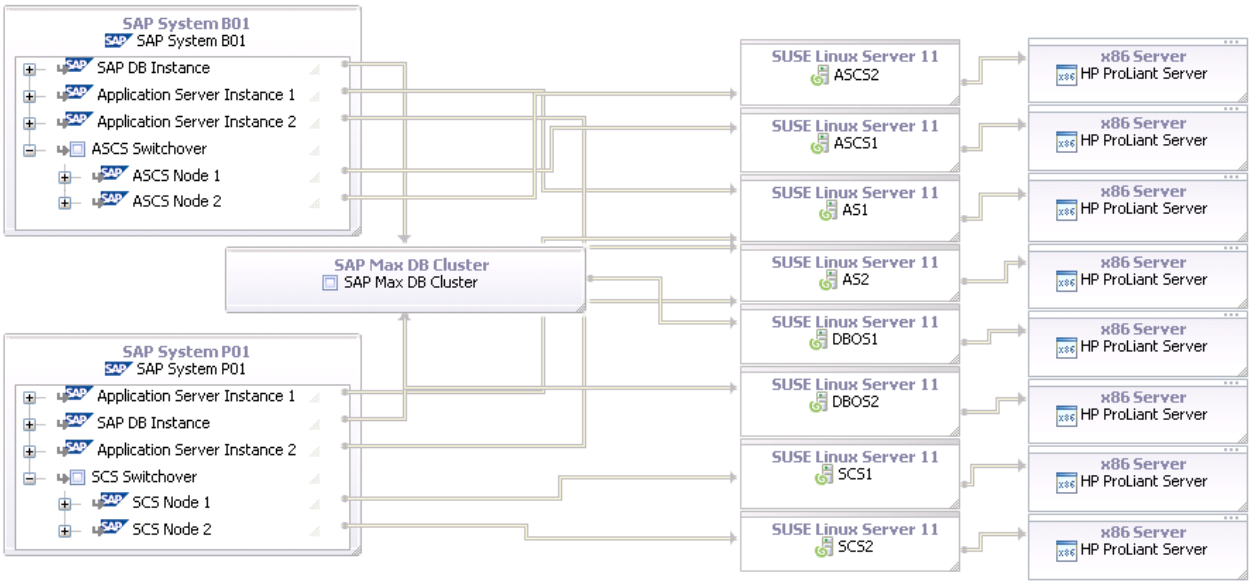

Figure 20: Example SAP system modelled as topology model

In the example topology model, shown in Figure 20, two SAP systems are modelled. For confidentiality reasons it is not further specified what particular SAP systems they represent. Naturally, big business applications consist of many interacting components, services, etc. These (sub-)units and the respective links are displayed in a tree view for clarity reasons but can also be displayed in the more common graph view. However, still visible as graph in the topology example are the links between SAP systems and the eight servers that are necessary to operate both systems.

Limitations

Even though modelling infrastructures of software systems and their interrelationship can be performed the IBM RSA has some limitations with regards to dependency analysis:

- RSA is intended to be used for development and modelling and only basic reasoning possibilities are supported.
- A second disadvantage is the extremely technical aspects that have to be modelled to have a comprehensive representation of the system.

Annex A.4 - Example Listing of CUDF for the TIMBUS Music Process in Taverna

preamble:

package: taverna-workbench

version: 2.3.0

architecture: all

depends: sun-java6-jre , graphviz , weka , somlib

conflicts: openjdk-6-jre

section: universe/science

package: graphviz

version: 2.26.3-5ubuntu4

architecture: amd64

maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>

depends: libc6 (>= 2.11), libcdt4, libcgraph5, libexpat1 (>= 1.95.8), libgd2-noxpm (>= 2.0.36~rc1~dfsg) | libgd2-xpm (>= 2.0.36~rc1~dfsg), libgraph4, libgvc5, libgvpr1, libx11-6, libxaw7, libxmu6, libxt6

recommends: ttf-liberation

suggests: gsfonts, graphviz-doc

conflicts: gdtclft

section: graphics

package: sun-java6-jre

version: 6.26-1ubuntu1

architecture: all

maintainer: Debian Java Maintainers <pkg-java-maintainers@lists.alioth.debian.org>

depends: debconf (>= 0.5) | debconf-2.0 , java-common (>= 0.24), locales, sun-java6-bin (>= 6.26-1ubuntu1) | ia32-sun-java6-bin (>= 6.26-1ubuntu1)

recommends: gsfonts-x11

suggests: sun-java6-plugin | ia32-sun-java6-plugin, sun-java6-fonts, ttf-baekmuk | ttf-unfonts | ttf-unfonts-core, ttf-kochi-gothic | ttf-sazanami-gothic, ttf-kochi-mincho | ttf-sazanami-mincho, ttf-arphic-uming

conflicts: j2se-common

replaces: ia32-sun-java6-bin, sun-java6-bin

section: partner/java

package: weka

version: 3.6.6

architecture: all

maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>

depends: openjdk-6-jre | sun-java6-jre, java-wrappers, cup (>= 0.11a+20060608)

Section: universe/science

package: somtoolbox

version: 0.7.5

architecture: all

maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>

depends: openjdk-6-jre | sun-java6-jre, java-wrappers, cup (>= 0.11a+20060608)

Section: universe/science

request:

install: taverna-workbench = 2.3.0

Annex A.5 – TIMBUS inverse relations mapping

Table 7: TIMBUS inverse constraint relation mappings

Name of relationship	Inverse relation
Requires	Asymmetric relationship
Conflicts	Symmetric relationship, Conflicts.
Pre-depends	Asymmetric relationship
Same-depends	Symmetric relationship, Same-depends
Post-depends	Asymmetric relationship

Table 8: TIMBUS non-exhaustive inverse description relation mappings

Name of relationship	Inverse relation
isA	isParentTo
isAssociationOf	hasAssociation
isDeliveryOf	hasDelivery
isExecutionOf	hasExecution
isFormatOf	hasFormat
isGuidanceOf	hasGuidance
IsLicenceOf	hasLicence
isPartOf	hasPart

Name of relationship	Inverse relation
isBelongingOf	hasBelonging
isConfigurationOf	hasConfiguration
IsNameOf	hasName
isTypeOf	hasType
isVendorOf	hasVendor
isVersionOf	hasVersion
IsSpecificationOf	hasSpecification
isSupporterOf	hasSupporter
Recommends/suggests	Provides

Annex A.6 – Listing of OWL-RDF properties of constraint relations

```
<Declaration>
   <ObjectProperty IRI="#hasProvider"/>
</Declaration>
<Declaration>
   <ObjectProperty IRI="#hasRequirement"/>
</Declaration>
<Declaration>
   <ObjectProperty IRI="#isConflictOf"/>
</Declaration>
<Declaration>
   <ObjectProperty IRI="#isRecommendationOf"/>
</Declaration>

<InverseObjectProperties>
   <ObjectProperty IRI="#isProviderOf"/>
   <ObjectProperty IRI="#hasProvider"/>
</InverseObjectProperties>
<InverseObjectProperties>
   <ObjectProperty IRI="#hasRequirement"/>
   <ObjectProperty IRI="#isRequirementOf"/>
</InverseObjectProperties>

<SymmetricObjectProperty>
   <ObjectProperty IRI="#hasConflict"/>
</SymmetricObjectProperty>

<SymmetricObjectProperty>
```

```
    <ObjectProperty IRI="#isConflictOf"/>
</SymmetricObjectProperty>

<AsymmetricObjectProperty>
    <ObjectProperty IRI="#hasProvider"/>
</AsymmetricObjectProperty>
```

7 References

1. Beagrie N., et al., 2001: Beagrie, N. and Jones, M. (2001) Preservation and Management of Digital Materials: A Handbook, The British Library, London. http://www.dpconline.org/advice/preservationhandbook
2. DPE 2007: DigitalPreservationEurope Partners, 2007: DPE Digital Preservation Research Roadmap. Public Deliverable D7.2, DPE.
3. Stevens, W.P, et al., 1974: Stevens, W.P.; Myers, G.J.; and Constantine, L.L. "Structured Design," IBM Systems Journal, 13:115-139, 1974.
4. IEEE 1471:2000: ISO/IEC/IEEE 42010 replaces IEEE 1471:2000. Recommended Practice for Architecture Description of Software-Intensive Systems. IEEE Computer Society Std. http://www.iso-architecture.org/ieee-1471
5. Sousa P., et al., 2006: Sousa, P., Caetano, A., Vasconcelos, A., Pereira, C. & Tribolet, J. (2006): Enterprise architecture modeling with the unified modeling language. In Enterprise Modeling and Computing with UML. IRM Press.
6. Aguilar-Savén R., 2004: R. Aguilar-Saven. Business process modelling: Review and framework. International Journal of Production Economics In Production Planning and Control, Vol. 90, No. 2. (28 July 2004), pp. 129-149, doi:10.1016/S0925-5273(03)00102-6
7. Safoora, K., et al., 2008: Safoora S. Khan and Greenwood, Phil and Garcia, Alessandro and Rashid, Awais. 2008. On the Impact of Evolving Requirements-Architecture Dependencies: An Exploratory Study. In Advanced Information Systems Engineering (LNCS). Springer Berlin / Heidelberg. ISBN: 978-3-540-69533-2. Pages: 243-257. Url: http://dx.doi.org/10.1007/978-3-540-69534-9_19
8. Norin J, 2007: Norin, Jens. 2007. LEAN Configuration Management Evolving the CM Discipline Through the Agile Paradigm Shift. Fall 2007 issue of Methods & Tools. http://www.methodsandtools.com/archive/archive.php?id=62
9. Bamford, R., et al., 1995: Robert Bamford, William J. Deibler II Software Systems Quality Consulting) -SSQC 1995 Configuration Management and ISO 9001 (http://www.ssqc.com/do25v6new.pdf)
10. CMU, 1995: Carnegie Mellon Univ. Software Engineering Inst. The Capability Maturity Model: Guidelines for Improving the Software Process, Addison Wesley, 1995.
11. Babich, W., 1986: Wayne A. Babich. (February 1986) Software Configuration Management: Coordination for Team Productivity.

12. Voelter, M., 2009: Best Practices for DSLs and Model-Driven Development. Journal of Object Technology, 8 (6), pp. 79-102.

13. Fritzsche, 2010: Performance related Decision Support for Process Modelling. PhD Thesis. School of Electronics, Electrical Engineering and Computer Science, Queens University Belfast.

14. Mattsson, S., et al., 1997: Mattsson, S. & Elmqvist, H (1997): Modelica - an international effort to design the next generation modeling language. In Proceedings of the 7th IFAC Symp. on Computer Aided Control Systems Design (CACSD'97)

15. ISO/IEC 2382-1:1993: ISO/IEC 2382-1:1993: ISO Information technology – Vocabulary, 1993. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=7229

16. ISO/IEC 2382-6:1987: ISO/IEC 2382-6:1987. Information processing systems -- Vocabulary -- Part 6: Preparation and handling of data. www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=7238

17. Kieburtz, R. et al., 1996: Kieburtz, R., McKinney, L., Bell, J., Hook, J., Kotov, A., Lewis, J., Oliva, D., Sheard, T., Smith, I. & Walton, L. (1996): A software engineering experiment in software component generation. In Proceedings of the 18th International Conference on Software Engineering (ICSE'96), pp. 542-552. IEEE Computer Society.

18. Jouault, F., et al., 2006: Jouault, F., Bezivin, J. & Kurtev, I. (2006): TCS: a DSL for the Specication of Textual Concrete Syntaxes in Model Engineering. In Proceedings of the 5th international conference on Generative programming and component engineering (GPCE'06), pp. 249-254. ACM.

19. Weller, S., et al., 1988: Weller, S., Romney, A. (1988). Systematic Data Collection (Qualitative Research Methods Series 10). Thousand Oaks, California: - Social Sciences information gathering

20. Weimer, J. (ed.), 1995: Research Techniques in Human Engineering. Englewood Cliffs, NJ

21. Buneman, P., et al., 1997: Peter Buneman, Susan Davidson, Mary Fernandez, and Dan Suciu. Adding Structure to Unstructured Data. In Proceedings of the International Conference on Database Theory, 1997.

22. Coxon A., 1999: Sorting Data: collection and analysis By Anthony Peter Macmillan Coxon -1999

23. Sapsford R., et al., 2006: Data Collection and Analysis By Dr. Roger Sapsford, Victor Jupp – 2006

24. TOGAF 2011: The Open Group (2011) TOGAF version 9.1. Van Haren Publishing

25. Archimate 2005: Lankhorst, M (2005) Enterprise Architecture at Work: Modelling, Communication, and Analysis. Springer, 2005.

26. Zachman, J., 1987: “A framework for information systems architecture,” IBM Systems Journal, vol. 12, no. 6, pp. 276–292.

27. OMG-BPMN, 2011: Object Management Group, Business Process Model and Notation (BPMN), Version 2.0, OMG Standard, formal/2011-01-03, 2011.

28. OAIS, 2002: CCSDS, Reference Model for an Open Archival Information System (OAIS) - Blue Book, 2002.

29. ISO14721:2003: ISO, Open archival information system – Reference model (ISO14721:2003), 2003.

30. Giaretta, D., 2007: The CASPAR Approach to Digital Preservation. The Internation Journal of Digital Curation, 2(1).

31. Conway, E., et al., 2011: Conway, E., Mattheus, B., Giaretta, D., Lambert, S., Draper, N. and Wilson, M. Managing Risks in the Preservation of Research Data with Preservation Networks, In the 7th International Digital Curation Conference, 2011.

32. OCLC, 2005: OCLC and RLG, Data Dictionary for Preservation Metadata, Final Report of the PREMIS Working Group. OCLC and RLG, 2005.

33. Szyperski, C. 2002: Szyperski, Clemens 2002 Component Software: Beyond Object-Orientated Programming (2nd Edition). Addison-Wesley Professional, 2 edition, November 2002.

34. Mancinelli F, 2006: Mancinelli, F.; Boender, J.; Di Cosmo, R.; Vouillon, J.; Durak, B.; Leroy, X.; Treinen, R., Managing the complexity of large free and open source package-based software distributions, Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference, pp 199-208,2006,IEEE.

35. Treinen, R., et al., 2008: Ralf Treinen, Stefano Zacchiroli. Description of the CUDF Format. CoRR Journal. Vol, abs/0811.3621. EE, http://arxiv.org/abs/0811.3621

36. OASIS, 2007: OASIS, Web Services Business Process Execution Language Version 2.0, OASIS Standard, 2007.

37. WSDL, 2007: World Wide Web Consortium, Web Services Description Language (WSDL) Version 2.0 Part I: Core Language, W3C Recommendation, 2007.

38. UML, 2007: Object Management Group, OMG Unified Model Language (OMG UML) Superstructure, Version 2.4.1, 2011.

39. SoaML, 2009: Object Management Group, Service oriented architecture Modeling Language (SoaML) – Specification for the UML Profile and Metamodel for Services (UPMS), OMG Adopted Specification, Finalisation Task Force Beta 2 document (FTF Beta 2), 2009.

40. Le, L. S., et al., 2010: Le, L. S., Ghose, A. K., Morrison, E., Definition of a Description Language for Business Service Decomposition, in Proceedings of the 1st International Conference on Exploring Services Sciences, Geneva, 2010.

41. ISO/IEC 20000-1, 2011: ISO/IEC 20000-1 - Service Management System standard, 2011

42. OGC-ITIL, 2007: Office of Government Commerce, The Official Introduction to the ITIL Service Lifecycle, The Stationery Office, 2007.

43. CMDB, 2010: Distributed Management Task Force, CMDB Federation (CMDBf) Frequently Asked Questions (FAQ) White Paper, Version 1.0.0, DMTF Informational, DSP2024, 2010.

44. IEEE: 828-2005: IEEE, IEEE Std 828-2005 – IEEE Standard for Software Configuration Management Plans, IEEE Computer Society, 2005.

45. ISO: 10007:2003: ISO, ISO 10007:2003 – Quality Management Systems – Guidelines for Configuration Management, 2003.

46. ISO: 9001:2008: ISO 9001:2008 - Quality management systems – Requirements, 2008

47. ANSI/EIA-649-2004: Electronic Industries Alliance (EIA), ANSI/EIA-649-2004 – National Consensus Standard For Configuration Management, 2004.

48. ISO/IEC 12207, 2008: ISO/IEC, ISO/IEC 12207 - Systems and software engineering - Software life cycle processes, International Organisation for Standardisation and International Electrotechnical Commission Std., 2008.

49. ISO/IEC 15288, 2008: ISO/IEC, ISO/IEC 15288 - Systems and software engineering - System life cycle processes, International Organisation for Standardisation and International Electrotechnical Commission Std., 2008.

50. IEEE-SWEBOK, 2004: IEEE Computer Society, Guide to the software engineering body of knowledge (swebok), IEEE, 2004 [Online]. Available: http://www.computer.org/portal/web/swebok

51. Sadiq, S., et al., 2007: Sadiq, S., Governatori, G., Naimiri, K. Modeling Control Objectives for Business Process Compliance. In Proceedings of the 5th International Conference on Business Process Management (BPM 2007), 2007.

52. Ghose, A., et al., 2007: Ghose, A., Koliadis, G. Auditing Business Process Compliance. In Proceedings of the 5th International Conference in Service Oriented Computing (ICSOC 2007), 2007.

53. Van der Aalst, et al., 2003: Van der Aalst, W. M. P., ter Hofstede, A.H.M., Weske, M. Business Process Management: A Survey. In Proceedings of the 1st International Conference on Business Process Management (BPM 2003), 2003.

54. Van der Aalst, et al., 2007: Van der Aalst, W. M. P. Challenges in Business Process Analysis. In Proceedings of the 9th International Conference on Enterprise Information Systems (ICEIS), 2007.

55. Van der Aalst, W. M. P., et al., 2007: Van der Aalst, W. M. P., Reijers, H. A., van Dongen, B. F., Alves de Medeiros, A. K., Song, M., Verbeek, H. M. W. Business Process Mining: An Industrial Application. In Information Systems 32, 713-732, 2007.

56. Uschold M., et al., 1996: Uschold M., King M., Moralee S., Zorgios Y. (1996), The Enterprise Ontology. AIAI, The University of Edinburgh. United Kingdom.

57. Fox M., et al., 1997: Fox M., Barbuceanu M., Gruninger M., Lin J. (1997) An Organisation Ontology for Enterprise Modeling. University of Toronto. Canada.

58. Krcmar, H., 2005: Informationsmanagement 4th ed. Heidelberg: Springer.

59. Leimeister, S. et al., 2010: The Business Perspective of Cloud Computing: Actors, Roles and Value Networks. In ECIS 2010 Proceedings. ECIS 2010 Proceedings. Available at: http://aisel.aisnet.org/ecis2010/56.

60. Knijff et al., 2011: Van der Knijff, J. and Wilson, C. (2011) Evaluation of Characterisation Tools - Part 1: Identification, Scape Project, FP7 ICT-2009.4.1-270137.

61. Rimal, B.P., et al., 2009: Rimal, B.P., Choi, E. & Lumb, I., 2009. A Taxonomy and Survey of Cloud Computing Systems. In 2009 Fifth International Joint Conference on INC, IMS and IDC. 2009 Fifth International Joint Conference on INC, IMS and IDC. IEEE, pp. 44–51.

62. Dumbill, E., 2012: What is big data?. http://radar.oreilly.com/2012/01/what-is-big-data.html?cmp=ba-conf-st12-twitter-promo, last visited: 2012-02-20

63. Steffen S., et al., 2009: Steffen Staab and Rudi Studer. 2009. Handbook on Ontologies (2nd ed.). Springer Publishing Company, Incorporated.

64. Gruber T. 1992: Tom R. Gruber. A translation approach to portable ontologies. Knowledge Acquisition, 5(2):199-220, 1993.

65. Uschold M., et al., 2006: Mike Uschold and Michael Gruninger, 2006. Ontologies: Principles, methods and applications. In Journal Knowledge Engineering Review, Vol. 11, pp93—136.