

Conversion and Emulation-aware Dependency Reasoning for the Needs of Digital Preservation

Ioannis Kargakis

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science

University of Crete

School of Sciences and Engineering

Computer Science Department

Voutes University Campus, Heraklion, GR-70013, Greece

Thesis Advisor: Assistant Prof. *Yannis Tzitzikas*

UNIVERSITY OF CRETE
COMPUTER SCIENCE DEPARTMENT

**Conversion and Emulation-aware Dependency Reasoning
for the Needs of Digital Preservation**

Thesis submitted by
Ioannis Kargakis
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: _____
Ioannis Kargakis

Committee approvals: _____
Yannis Tzitzikas
Assistant Professor, Thesis Supervisor

Dimitris Plexousakis
Professor, Committee Member

Anastasia Analyti
Principal Researcher, Committee Member

Departmental approval: _____
Angelos Bilas
Professor, Director of Graduate Studies

Heraklion, December 2013

Conversion and Emulation-aware Dependency Reasoning for the Needs of Digital Preservation

Abstract

Modern society and economy is increasingly dependent on a deluge of only digitally available information. Its preservation within an unstable and rapidly evolving technological (and social) environment is a challenging problem of prominent importance. Digital material has to be preserved not only against loss or corruption, but also against hardware/software technology changes, plus changes in the knowledge of the community.

In this thesis we propose a modeling and reasoning approach for modeling the dependencies of the digital material to be preserved. It is based on an object-oriented model, enriched with rules, while a distinctive feature of the approach is that it allows modeling converters and emulators (indispensable in most preservation strategies) and considers their capabilities in the offered reasoning services. We show that the proposed modeling achieves the desired reasoning, thus enables offering more advanced digital preservation services which can reduce the human effort required for checking the performability of tasks over digital objects, for predicting the consequences of probable losses or obsolesces, and can assist preservation planning. The thesis provides examples demonstrating how real world converters and emulators can be modeled, and describes how the approach can be implemented in various technologies, as well as how it can be applied and exploited in general. In the sequel the thesis details the demonstrator called “Epimenides”, which is based on Semantic Web technologies, and realizes the proposed approach and thus proves its feasibility. Its knowledge base already contains the MIME types and their associated tasks. Finally various promising evaluation results are reported.

Συλλογιστική Εξαρτήσεων με Υποστήριξη Μετατροπών και Εξομοιωτών για τις ανάγκες της Ψηφιακής Διατήρησης

Περίληψη

Η σύγχρονη κοινωνία και οικονομία όλο και περισσότερο εξαρτάται από πληροφορίες που είναι διαθέσιμες μόνο σε ψηφιακή μορφή. Η διατήρησή της (preservation) σε ένα ταχέως εξελισσόμενο τεχνολογικό και κοινωνικό περιβάλλον αποτελεί πρόκληση μείζονος σημασίας. Το ψηφιακό υλικό πρέπει να προστατευθεί όχι μόνο από απώλεια ή φθορά (corruption), αλλά και από τις αλλαγές στο υλικό, το λογισμικό, ή τη γνώση της κοινωνίας, ώστε να παραμείνει καταληπτό και λειτουργικό.

Σε αυτήν τη διατριβή προτείνουμε μια προσέγγιση μοντελοποίησης και συλλογισμού η οποία επιτρέπει τη μοντελοποίηση των εξαρτήσεων του ψηφιακού υλικού που θέλουμε να διατηρήσουμε. Βασίζεται σε ένα αντικειμενοστρεφές εννοιολογικό μοντέλο, εμπλουτισμένο με κανόνες, ενώ ένα ιδιαίτερο χαρακτηριστικό της προσέγγισης είναι ότι επιτρέπει τη μοντελοποίηση μετατροπών (converters) και εξομοιωτών (emulators) (οι οποίοι είναι απαραίτητοι στις περισσότερες στρατηγικές διατήρησης), και μάλιστα αξιοποιεί τις δυνατότητές τους στις προσφερόμενες συμπερασματικές διαδικασίες. Δείχνουμε ότι ο προτεινόμενος τρόπος μοντελοποίησης επιτυγχάνει την απαιτούμενη αυτόματη συλλογιστική και καθιστά εφικτή την παροχή προηγμένων υπηρεσιών ψηφιακής διατήρησης οι οποίες μπορούν να μειώσουν σημαντικά την ανθρώπινη προσπάθεια που απαιτείται για τον έλεγχο της επιτευξιμότητας εργασιών επί του ψηφιακού υλικού, την πρόβλεψη συνεπειών από απώλεια ή παρώχηση, κ.α., και να βοηθήσουν τη διαδικασία του προγραμματισμού διατήρησης (preservation planning). Η διατριβή παρέχει παραδείγματα που επιδεικνύουν πως πραγματικοί μετατροπείς και εξομοιωτές μπορούν να μοντελοποιηθούν, περιγράφει τρόπους υλοποίησης της προσέγγισης σε διαφορετικές τεχνολογίες, και γενικότερα προτείνει τρόπους εφαρμογής και αξιοποίησης αυτής της προσέγγισης. Στη συνέχεια περιγράφεται το σύστημα επίδειξης «Επιμενίδης», το οποίο βασίζεται σε τεχνολογίες του Σημαιολογικού Ιστού, που πραγματώνει την προσέγγιση μοντελοποίησης και συλλογισμού, και εκ τούτου την επαληθεύει. Η βάση γνώσεων του ήδη περιλαμβάνει τους τύπους MIME και των συναφών εργασιών. Τέλος παρουσιάζονται διάφορα πολύ θετικά αποτελέσματα αξιολόγησης.

Ευχαριστίες

Στο σημείο αυτό θα ήθελα να ευχαριστήσω τον επόπτη καθηγητή μου κ. Γιάννη Τζίτζικα για την ορθή καθοδήγηση και ουσιαστική συμβολή του στην ολοκλήρωση της παρούσας διατριβής. Επιπλέον, να εκφράσω τις ευχαριστίες μου στον κ. Δημήτρη Πλεξουσάκη και στην κ. Αναστασία Αναλυτή για την προθυμία τους να συμμετέχουν στην τριμελή επιτροπή.

Ακόμα να ευχαριστήσω το Ινστιτούτο Πληροφορικής του Ιδρύματος Τεχνολογίας και Έρευνας για την πολύτιμη υποστήριξη σε υλικοτεχνική υποδομή και τεχνογνωσία, καθώς και για την υποτροφία που μου προσέφερε καθ' όλη τη διάρκεια της μεταπτυχιακής μου εργασίας.

Πολλές ευχαριστίες σε όλους τους φίλους μου για την στήριξη, την εμπύχωση τους, καθώς και για όλες τις όμορφες στιγμές που μοιραστήκαμε.

Το μεγαλύτερο ευχαριστώ όμως αξίζει οι γονείς μου Αντώνης και Κατερίνα, και ο αδερφός μου Γιώργος για την υποστήριξη, την συμπαράσταση και την εμπιστοσύνη που μου έδειξαν όλα αυτά τα χρόνια των σπουδών μου, αλλά και για την αγάπη τους σε κάθε βήμα της ζωής μου. Σας ευχαριστώ πολύ για όλα.

στους γονείς μου

Contents

1	Introduction	3
2	Related Work, Background & Requirements	7
2.1	What is Digital Preservation?	7
2.2	Related Work: Dependency Management for Digital Preservation .	10
2.3	Background: Migration and Emulation	12
2.3.1	The KEEP Project	13
2.4	Background: Datalog	14
2.4.1	Example (of Datalog-like modeling)	15
2.5	Requirements on Reasoning Services	16
3	Modeling Tasks and their Dependencies	19
3.1	Overview of the Modeling Approach	19
3.2	General Methodology	20
3.3	Modeling Digital Objects, Type Hierarchies, and Profiles	21
3.4	Modeling Task-Dependencies and Task Hierarchies	22
3.5	Modeling Converters	23
3.6	Modeling Emulators	24
3.6.1	Modeling Important Parameters	26
3.6.2	Handling Exceptions or Special Cases	27
3.7	Modeling Real Converters and Emulators	27
3.8	Synopsis of the Modeling Approach	29
4	Reasoning Services	31
4.1	Task-Performability	31
4.2	Consequences of a Hypothetical Loss	32

4.3	Computation of Gaps (Missing Modules)	32
5	Implementation	35
5.1	Possible Implementation Technologies	35
5.2	On RDF/S-based Implementations	36
5.2.1	Modules	37
5.2.2	Profiles	38
5.2.3	Dependencies	38
5.2.4	Converters	39
5.2.5	Emulators	39
5.2.6	Services	40
5.3	Evolution, Representation of Profiles and Limitations	40
5.3.1	Inference and Evolution	40
5.3.2	Representation of Profiles	41
5.3.3	Limitations of RDF/S	42
5.4	Proof-of-Concept Dataset and Repository	43
6	Epimenides: A Proof-of-Concept System	45
6.1	Use Cases	45
6.2	Deployment of Epimenides	46
6.3	User Interface	47
6.4	The Knowledge Base of Epimenides	48
6.4.1	The Current Knowledge Base	53
6.5	Aiding the Ingestion of Tasks	54
6.6	Aiding the Ingestion of Converters and Emulators	54
6.7	Screen Dumps of Epimenides	55
6.8	Evaluating its Usability	60
6.9	Query and Reasoning Efficiency	61
7	Applicability	65
7.1	On Applicability	65
7.2	Ways to Offer the Dependency Management Approach	66
7.3	Application by Extending an Existing Repository (Fedora)	67
7.4	Related Datasets and Tools	68
7.4.1	PreScan	68

7.4.2	The PRONOM Registry and its Contents	69
7.4.3	Catalogues of Existing Converters and Emulators	71
7.5	Case Study: DANS	71
7.5.1	Scenario 1: Checking File Format Compatibility (compliance or migratability) with Acceptable/Preferred File Formats during Ingestion	72
7.5.2	Scenario 2: Updating the List of Preferred/Acceptable Formats and Detecting the Consequences of Obsolete Formats	73
7.5.3	Scenario 3: Assistance in Planning and Performing Migration to Acceptable/Preferred File Formats	74
7.5.4	Scenario 4: Checking the Preservation of the Software	75
7.5.5	Scenario 5: Bit Preservation (ability to test corruption)	75
7.5.6	Consolidation of the Scenarios	76
7.6	Layering Tasks	76
8	Concluding Remarks	79
	Bibliography	80
	Appendix A	87
A.1	Epimenides: RDF Schema	87

List of Figures

1.1	Running example. (a)The situation (b)The profile (c)A series of conversion/emulation to achieve our objective	4
2.1	A Curation Lifecycle Model	9
2.2	Preservation system	11
3.1	Informal concept map	19
4.1	The proof tree of the running example	32
5.1	Exploration System	44
6.1	Use Case Diagram of Epimenides	46
6.2	The deployment diagram of Epimenides	47
6.3	Main functionality of Epimenides	47
6.4	Checking the performability over a digital object	48
6.5	Architecture of the KB	49
6.6	The contents of an RDF/S KB that follows the architecture of Figure 6.5	50
6.7	Operational KB	51
6.8	Gradual Expansion	52
6.9	The Gradual Expansion in Epimenides	53
6.10	Load your personal profile or use a demo profile	56
6.11	Upload digital objects to check the performability of them	56
6.12	System finds the tasks that usually make sense to apply to the uploaded digital objects	57
6.13	Results of analysis	57

6.14 Exploring the Dependencies of a Task	58
6.15 Identify the modules that will be affected on a task after removing a module	58
6.16 Define a new Task	59
6.17 Define a new Emulator Type	59
6.18 Explore the contents of the underling RDF/S triple store	60
6.19 Analysis of the responses to the questionnaire	62
7.1 Our approach in a software example	66
7.2 The system PreScan	68

List of Tables

2.1	Modeling the running examples with Facts and Rules	16
3.1	Facts of running examples	21
5.1	Implementation Approaches	36
6.1	Some indicative measurements of time	63
7.1	The RDF properties of PRONOM	70
7.2	Application of the Methodology for the case of DANS	77

Chapter 1

Introduction

Today the majority of the information exists in digital form (such as pdf or doc files, emails, blogs, videos, social networking websites etc.) while a few decades ago the information existed only in physical form (written in stone, paper, papyrus, wood etc.). The preservation of the information has been always a major issue. The information that was stored in physical materials should be protected from natural disasters (fire, flood, earthquakes), while today a set of activities (migration, emulation, metadata attachment etc.) is required to preserve the digital information. These activities refer to the term of the “Digital Preservation”.

Digital material has to be preserved not only against loss or corruption, but also against hardware/software technology changes, plus changes in the knowledge of the community. Consequently there is a need for services that help archivists in checking whether the archived digital artifacts remain *intelligible* and *functional*, and in identifying the consequences of probable losses (obsolescence risks).

Past works [1], [2] and [3] have shown how the above mentioned services can be approached from a *dependency management* perspective. However, the aforementioned works did not capture *converters* and *emulators*. Since conversion (or migration) and emulation are quite important preservation strategies, a dependency management approach should allow modeling explicitly converters and emulators (and analyze them from a dependency point of view, since they have to be preserved too), and exploit them during the offered preservation services.

This is important since a sequence of conversions and emulations can be enough for vanishing an intelligibility gap, or for allowing performing a task. Note that

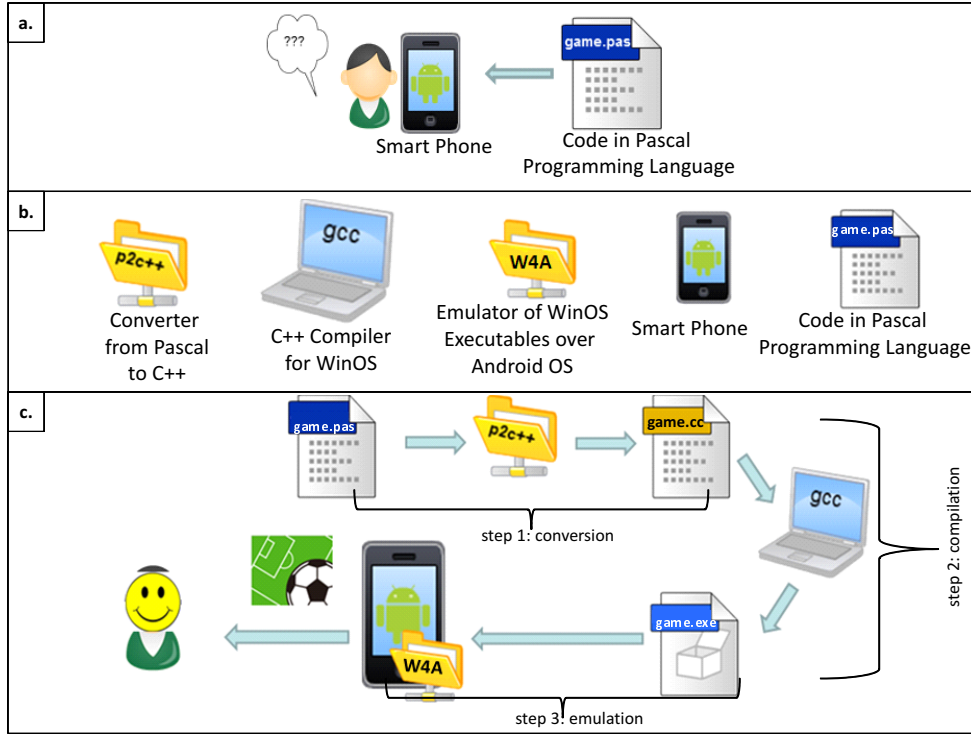


Figure 1.1: Running example. (a)The situation (b)The profile (c)A series of conversion/emulation to achieve our objective

there is a plethora of emulation and migration approaches and tools that concern various layers of a computer system (from hardware to software), or various source/target formats (e.g. see [4] for an overview). This means that it is beneficial to use advanced knowledge management techniques for aiding the exploitation of all possibilities that the existing and emerging emulators/converters enable, and assist *preservation planning* (e.g. [5]). This is crucial since the scale and complexity of information assets and systems evolve towards overwhelming the capability of human archivists and curators (either system administrators, programmers and designers).

Below we attempt to pass the main message through an example. Consider a user, say James, who would like to run on his mobile phone, software source code written before many years, e.g. software code written in Pascal programming language, stored in a file named `game.pas`. For example consider the situation illustrated in Figure 1.1a.

The rising questions are:

- What can James do? (to achieve his objective)

- What should we (as community) do?
 - Do we have to develop a Pascal compiler for Android OS?
 - Do we have to standardize programming languages?
 - Do we have to standardize operating systems, virtual machines, and so on?

The direction and answer (according to this thesis), is that it is worth investigating whether it is already possible to run that code on android by “combining” existing software, i.e. by applying a series of transformations and emulations.

To continue this example, suppose that we have in our disposal only the following (as shown in Figure 1.1b):

- a converter from Pascal source code to C++ source code (say `p2c++`),
- a C++ compiler (`gcc`) for Windows OS,
- an emulator of Windows OS executable over Android OS (say `W4A`).
- a smart phone running Android OS
- a Pascal File (`game.pas`)

Someone could then think that it seems that we could run `game.pas` on his mobile phone in three steps : (step 1) by first converting the Pascal code to C++ code, (step 2) then compiling the C++ code to produce executable code, and finally (step 3) by running over the emulator the executable yielded by the compilation. Indeed, the series of transformation/emulations shown in Figure 1.1c could achieve our objective.

One might argue that this is very complex for humans. Indeed this is true. We believe that such reasoning should be done by computers, not humans. The work that we present in the current thesis shows how we can model our information in a way that enables this kind of *automated reasoning*.

The above scenario concerns software. We should however clarify that the proposed approach is not confined to software. Various services that concern documents and datasets can also be captured.

In a nutshell, the main contributions of this thesis are: (a) we extend past dependency management approaches for digital preservation with converters and emulators, (b) we demonstrate how this modeling apart from capturing the preservability of converters and emulators, enables the desired reasoning regarding task

performability, risk detection etc, (c) we show that with this approach we can model real converters and emulators, (d) we discuss implementation approaches and detail a particular one which we implemented using recently emerged Semantic Web tools, and (e) we present the prototype system **Epimenides** that realizes this approach. Furthermore we discuss how the proposed functionality can be applied or injected to existing systems.

The rest of this thesis is organized as follows: Chapter 2 discusses the motivation, the context, past related works, the required background, and the key requirements. Chapter 3 introduces the rule-based modeling, and provides examples demonstrating how real converters and emulators can be modeled. Chapter 4 discusses how the corresponding inference services can be realized using the proposed modeling, and Chapter 5 shows how the approach can be implemented using Semantic Web tools. Chapter 6 describes the proof-of-concept system **Epimenides**, details its implementation which is founded on Semantic Web technologies, and reports various results. Chapter 7 discusses methods to apply the dependency management approach, methodological issues, as well as a case study. Finally Chapter 8 summarizes, discusses related issues and identifies issues for further research.

Chapter 2

Related Work, Background & Requirements

2.1 What is Digital Preservation?

The term “Digital Preservation” refers to the series of managed activities required to ensure continued access to digital information for as long as necessary [6]. Generally we can distinguish the digital preservation in 3 terms:

- **Long-term preservation.** Continued access to digital materials, or at least to the information contained in them, indefinitely.
- **Medium-term preservation.** Continued access to digital materials beyond changes in technology for a defined period of time but not indefinitely.
- **Short-term preservation.** Access to digital materials either for a defined period of time while use is predicted but which does not extend beyond the foreseeable future and/or until it becomes inaccessible because of changes in technology

The digital information needs the continuous management to prevent problems that can arise from the technological change due to the passage of time and this is where the Digital Preservation Systems can contribute. Physical storage media, data formats, hardware, and software all become obsolete over time, posing significant threats to the survival of the content of a digital object. Because of this

digital obsolescence, where a digital resource is no longer functional, the digital preservation process of a resource should start as early in the lifecycle (even in the creation) of this digital resource as possible.

Digital preservation today is a major procedure as more and more individuals and organizations create new digital information, or even they digitize the non-digital materials that hold. For example private users wants to keep accessible their photo, audio, or video collections while insurance and aviation companies, the pharmaceutical and car industry, and other key players want to preserve their data holdings, simulation models, or studies over time [7].

Amongst the many strategies developed to preserve digital objects and keep them accessible in the long run, according to Wikipedia¹ the following are the most prominent :

- **Refreshing**, is the transfer of data between two types of the same storage medium so there are no bitrot changes or alteration of data. For example, transferring census data from an old preservation CD to a new one.
- **Migration** is the transferring of data to newer system environments (more about migration in Section 2.3).
- **Emulation** is the replicating of functionality of an obsolete system (more about migration in Section 2.3).
- **Replication** is the creation of duplicate copies of data on one or more systems.
- **Encapsulation.** This method maintains that preserved objects should be self-describing, virtually “linking content with all of the information required for it to be deciphered and understood”.
- **Metadata attachment.** Metadata is data on a digital file that includes information on creation, access rights, restrictions, preservation history, and rights management. Metadata attached to digital files may be affected by file format obsolescence.

To standardize digital preservation practice and provide a set of recommendations for preservation program implementation, the Reference Model for an *Open*

¹http://en.wikipedia.org/wiki/Digital_preservation#Strategies

Archival Information System (OAIS) was developed. OAIS is an ISO reference (ISO 14721:2003) defined by a recommendation of the Consultative Committee for Space Data Systems. Is concerned with all technical aspects of a digital object's life cycle: ingest, archival storage, data management, administration, access and preservation planning. The model also addresses metadata issues and recommends that five types of metadata be attached to a digital object: reference (identification) information, provenance (including preservation history), context, fixity (authenticity indicators), and representation (formatting, file structure, and what “imparts meaning to an object's bitstream”).

The Digital Curation Centre² (DCC) introduces a graphical model³ that provides a high level overview of the stages required for successful curation and preservation of data from initial conceptualisation through the iterative curation cycle.

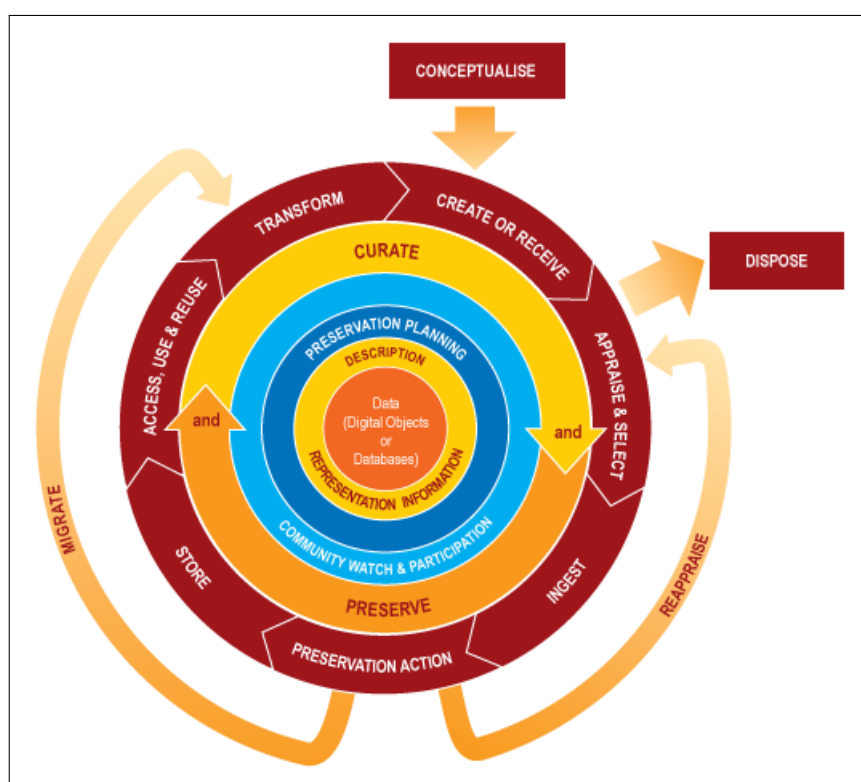


Figure 2.1: A Curation Lifecycle Model

²<http://www.dcc.ac.uk/>

³<http://www.dcc.ac.uk/sites/default/files/documents/publications/DCCLifecycle.pdf>

Figure 2.1 shows this model. The key elements that the Curation Lifecycle contains are : (a) the Data (Digital Objects, Databases), (b) the Full Lifecycle Actions (Description and Representation Information, Preservation Planning, Community Watch and Participation, Curate and Preserve), (c) the Sequential Actions (Conceptualise, Create or Receive, Appraise and Select, Ingest, Preservation Action, Store, Access, Use and Reuse, Transform) and (d) the Occasional Actions (Dispose, Reappraise, Migrate). The model can be used to plan activities within a specific research project, organisation, or consortium to ensure all necessary stages are undertaken, each in the correct sequence. It is important to note that the description, preservation planning, community watch, and curate and preserve elements of the model should be considered at all stages of activity.

The rising question in a digital preservation system is: *what should we preserve and how?* Certainly, we have to preserve the file bitstreams of digital objects. However we should also try to preserve their intelligibility and functionality, in such a way that allows the performing of tasks (e.g run, render, compile etc.) over a digital object, and this is the main aspect that we elaborate in this thesis.

2.2 Related Work: Dependency Management for Digital Preservation

As sketched in Figure 2.2, a preservation system consists of the stored digital material (to be preserved), plus a number of services for managing the curating this material. Many of these services rely on metadata and in some cases on other external resources and services. Some basic preservation services can be reduced to dependency management services, and a semantic registry can be used for offering a plethora of curation services.

Intelligibility checking is a very fundamental service for digital preservation. [1] showed how this service can be reduced to *dependency management* services, and how a semantic registry (compatible with OAIS) can be used for offering a plethora of curation services. Subsequently, [2] extended that model with *disjunctive dependencies*. The key notions of these works is the notion of *module*, *dependency* and *profile*. In a nutshell, a *module* can be a software/hardware component or even a knowledge base expressed either formally or informally, explicitly or tacitly, that

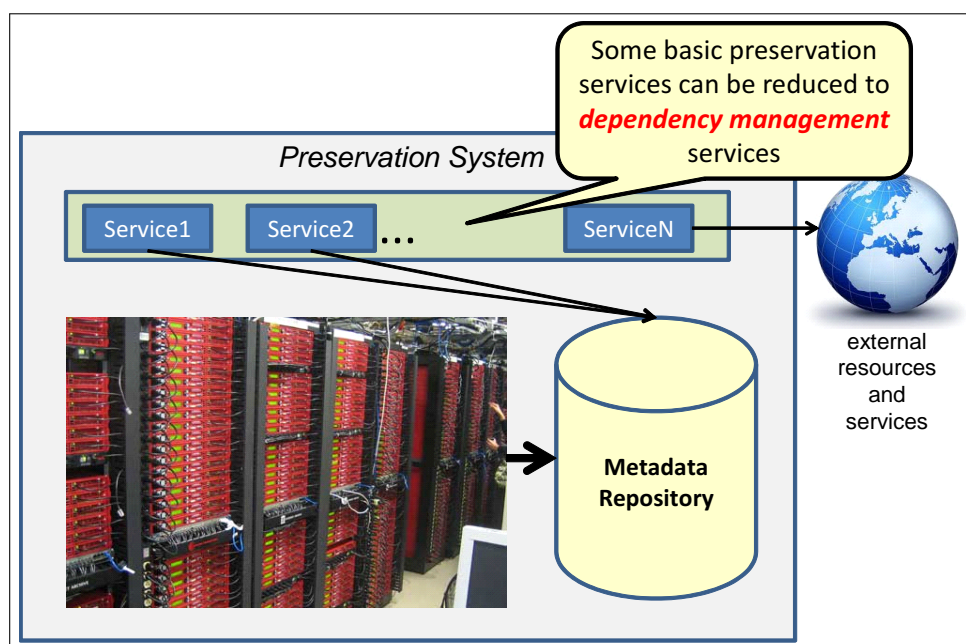


Figure 2.2: Preservation system

we want to preserve. A module may require the availability of other modules in order to function, be understood or managed.

A *profile* is the set of modules that are assumed to be known (available or intelligible) by a user (or community of users), and this notion allows controlling the number of dependencies that have to be recorded formally (or packaged in the context of an *encapsulation preservation strategy*). Subsequently, and since there is not any objective method to specify exactly which are the dependencies of a particular digital object, [8] extended the model with *task-based* dependencies where the notion of *task* is used for determining the dependencies of an object. As *tasks* we define actions that can be applied on a digital object (e.g. edit, render or run a digital object). In [8] actually introduced an extensible *object-oriented* modeling of dependency graphs expressed in Semantic Web (SW) languages (RDF/S). Based on that model, a number of services were defined for checking whether a module is *intelligible* by a community (or for computing the corresponding *intelligibility gap*), or for checking the *performability of a task*. These dependency management services were realized over the available SW query languages. For instance, GapMgr⁴

⁴ <http://athena.ics.forth.gr:9090/Applications/GapManager/>

and PreScan⁵ [9] are two systems that have been developed based on this model, and have been applied successfully in the context of the EU project CASPAR⁶. Subsequently, [3] introduced a *rule-based* model which also supports task-based dependencies, and (a) simplifies the disjunctive dependencies of [2], and (b) is more expressive and flexible than [8] as it allows expressing the various properties of dependencies (e.g. transitivity, symmetry) straightforwardly. That work actually reduced the problem of dependency management to *Datalog*-based modeling and query answering.

However, none of the aforementioned works were able to model and manage *converters* and *emulators*.

2.3 Background: Migration and Emulation

Migration is the process of converting a digital object that runs on one platform so that it will run on another (non-obsolete) platform [10]. Its purpose is to preserve the integrity of digital objects and to retain the ability for clients to retrieve, display, and otherwise use them in the face of a constantly changing technology [11].

Emulation is generally described as imitating a certain computer platform or program on another platform or program (for a discussion see [12, 10]). It requires the creation of *emulators*, where an emulator is hardware or software or both that duplicates (or emulates) the functions of a first computer system (the guest) in a different second computer system (the host), so that the emulated behavior closely resembles the behavior of the real system. Popular examples of emulators include QEMU [13], Dioscuri [14], etc. There is currently a rising interest on emulators for the needs of digital preservation [15]. Just indicatively, [16] overviews the emulation strategies for digital preservation and discusses related issues, and several recent projects have focused on the development of emulators for the needs of digital preservation (e.g. see [14] and [17], while [14] compares applications running on Dioscuri with the same applications executed directly on the host machine).

Another related concept is that of the Universal Virtual Computer (UVC) that was introduced in [18] (more recent in work [19]). It is a special form of Emulation where a hardware and software independent platform is implemented, where files

⁵ <http://www.ics.forth.gr/isl/PreScan>

⁶ <http://www.casparpreserves.eu/>

are migrated to UVC internal representation format and where the whole platform can be easily emulated on newer computer systems. It is like an intermediate language for supporting emulation.

In brief, and from a dependency perspective, we could say that the *migration* process *changes the dependencies* (e.g. the original digital object depends on an old format, while the migrated digital object now depends on a newer format). Regarding *emulation* we could say that the emulation process does not change the “native” dependencies of digital objects. An emulator essentially makes available the behavior of an old module (actually by emulating its behavior). It follows that the availability of an emulator can “satisfy” the dependencies of some digital objects (as described in the running example of `game.exe` in Chapter 1), but we should note that the emulator itself (`w4a` in the running example) has its own dependencies that have to be preserved to ensure its performability (this will be made evident in Section 3.6). The same also holds for converters.

2.3.1 The KEEP Project

We could also mention here the KEEP⁷ (Keeping Emulation Environments Portable) project that aims at developing emulation services to enable accurate rendering of both static and dynamic digital objects. The overall aim of the project is to facilitate universal access to cultural heritage resources. KEEP has created an Emulation Framework⁸ [20] (EF) which provides additional services which will help to build a more solid ground for the emulation preservation strategy. KEEP is depending on existing and future emulators, and has not created an emulator itself.

The EF offers a convenient way to render digital files and programs in their native computer environment. It offers users the potential to view these files in their intended look and feel, independent from current state of the art computer systems. The Emulation Framework automatically selects and runs the best available emulator and configures the software dependencies required to render the object (operating system, applications, etc.).

The EF contains the Emulator and the Software Archive. The Emulator Archive database holds the binaries and metadata for the available emulators. The Software Archive has been created to manage the software required by the

⁷<http://www.keep-project.eu/ezpub2/index.php?/eng>

⁸<http://emuframework.sourceforge.net/>

emulators. The Software Archive defines a Pathway in an XSD schema. Pathway is called the environment that can render digital objects, consisting of the digital object file format, and hardware platform and possibly an application and/or operating system.

Each EF-compliant emulator is transferred from the Emulator Archive to a receiver in an Emulator Package XSD Schema. This schema describes the emulator software and includes some descriptive fields (such as name, version, and description) and technical elements such as a list of hardware that the emulator can emulate, a list of software image format (such as FAT12, FAT32, D64, etc.) that the emulator can read. Also contains information about the executable itself.

The downloaded package contains the emulators (in the Emulator Archive) :

- Dioscuri - x86 Java-based emulator capable of running MS DOS and Linux;
- QEMU - x86 capable of running MS Windows and Linux;
- VICE - Commodore 64 emulator;
- UAE - Amiga emulator;
- Java CPC - Amstrad emulator;
- BeebEm - BBC Micro emulator;
- Thomson T07 - Thomson T07 emulator;

One can also add an emulator via the GUI of the EF, but he has to define all the parameters manually.

2.4 Background: Datalog

We will base our modeling and reasoning approach in Datalog [21] which is a query and rule language for deductive databases (syntactically subset of Prolog). In brief, a Datalog program consists of *facts*, e.g. `JavaFile(myfile.java)`, and *rules*. An example of a rule having a *head* with a predicate of two variables and a *body* with two monadic predicates is: `Compilable(X,Y) :- JavaFile(X), JavaCompiler(Y)`, which is read as follows: if we have a javafile *f1*, and a java compiler *f2*, then we can infer that *f1* is compilable by *f2*. In Datalog, the set of predicates is partitioned into two disjoint sets, *EPred* and *IPred*. The elements of *EPred* denote extensionally defined predicates, i.e. predicates whose extensions are given by the facts of the Datalog programs (i.e. tuples of database tables),

while the elements of *IPred* denote intensionally defined predicates, where the extension is defined by means of the rules of the Datalog program.

In our context, the proposed implementation will be described in Chapter 5.

2.4.1 Example (of Datalog-like modeling)

James has a laptop where he has installed the `NotePad` text editor, the `javac 1.6` compiler for compiling Java programs and `JRE1.5` for running Java programs (bytecodes). He is learning to program in Java and C++ and to this end, and through `NotePad` he has created two files, `HelloWorld.java` and `HelloWorld.cc`, the first being the source code of a program in java, the second of one in C++. Consider another user, say Helen, who has installed in her laptop the `Vi` editor and `JRE1.5`.

Suppose that we want to preserve these files, i.e. to ensure that in future James and Helen will be able to edit, compile and run these files. In general, to edit a file we need an editor, to compile a program we need a compiler, and to run the bytecodes of a Java program we need a Java Virtual Machine. To ensure preservation we should be able to express the above.

To this end we could use facts and rules. For example, we could state: *A file is editable if it is TextFile and a TextEditor is available*. Since James has two text files (`HelloWorld.java`, `HelloWorld.cc`) and a text editor (`NotePad`), we can conclude that these files are editable by him. By a rule of the form: *If a file is Editable then it is Readable too*, we can also infer that these two files are also readable. We can define more rules in a similar manner to express more task-based dependencies, such as compilability, runability etc. For our running example we could use the facts and rules which are describing in table 2.1.

The last two columns indicate which facts are valid for James and which for Helen. From these we can infer that James is able to compile the file `HelloWorld.java` and that if James sends his `TextFiles` to Helen then she can only edit them but not compile them since she has no facts about Compilers.

Let us now extend our example with *converters* and *emulators*. Suppose James has also an old source file in Pascal PL, say `game.pas`, and he has found a *converter* from Pascal to C++, say `p2c++`. Further suppose that he has just bought a smart phone running Android OS and he has found an *emulator* of WinOS over Android

Facts and Rules	James	Hellen
Facts		
NotePad is a TextEditor	✓	
VI is a TextEditor		✓
HelloWorld.java is a JavaFile	✓	
HelloWorld.cc is a C++File	✓	
javac1.6 is a JavaCompiler	✓	
JRE1.5 is a JVM	✓	✓
gcc is a C++Compiler	✓	
Rules		
A file is Editable if it is a TextFile and a TextEditor is available		
A file is JavaCombilable if it is a JavaFile and a JavaCompiler is available		
A file is C++Combilable if it is a C++File and a C++Compiler is available		
A file is Compilable if it is JavaCompilable or C++Compilable		
A file is a TextFile if it is JavaFile or C++File		
If a file is Editable then it is Readable		

Table 2.1: Modeling the running examples with Facts and Rules

OS. It should follow that James can run `game.pas` on his mobile phone (by first converting it in C++, then compiling the outcome, and finally by running over the emulator the executable yielded by the compilation).

2.5 Requirements on Reasoning Services

Regarding curation services, we have identified the following key requirements :

Task-Performability Checking. To perform a task we have to perform other subtasks and to fulfil associated requirements for carrying out these tasks. Therefore, we need to be able to decide whether a task can be performed by examining all the necessary subtasks. For example, we might want to ensure that a file is runnable, editable or compilable. This should also exploit the possibilities offered by the availability of converters. For example, the availability of a converter from Pascal to C++, a compiler of C++ over Windows OS and an emulator of Windows OS over Android OS should allow inferring that the particular Pascal file is runnable over Android OS.

Consequences of a Hypothetical Loss. The loss or removal of a software module could also affect the performability of other tasks that depend on it and thus break a chain of task-based dependencies. Therefore, we need to be able to

identify which tasks are affected by such removals.

Identification of missing resources to perform a task. When a task cannot be carried out it is desirable to be able to compute the resources that are missing. For example, if Helen wants to compile the file `HelloWorld.cc`, her system cannot perform this task since there is not any C++Compiler. Helen should be informed that she should install a compiler for C++ to perform this task.

Support of Task Hierarchies. It is desirable to be able to define task-type hierarchies for gaining flexibility, supporting various levels of granularity, and reducing the number of rules that have to be defined.

Properties of Dependencies. Some dependencies are *transitive*, some are not. Therefore we should be able to define the properties of each kind of dependency.

Here we should clarify that we do not focus on modeling, logging or reasoning over *composite tasks* in general (as for example it is done in [22]). We focus on the requirements for ensuring the performability of simple (even atomic) tasks, since this is more aligned with the objectives of long term digital preservation. Neither we focus on modeling or logging the particular workflows or derivation chains of the digital artifacts, e.g. using *provenance* models like OPM or CRM Dig [23]. We focus only on the dependencies for carrying out the desired tasks. Obviously this view is less space consuming, e.g. in our running example we do not have to record the particular compiler that was used for the derivation of an executable (and its compilation time, or who achieved the compilation), we just care to know what compiler one needs to have for future use. However, if a detailed model of the process is available, then the dependency model can be considered as a more simple view of that model.

Chapter 3

Modeling Tasks and their Dependencies

3.1 Overview of the Modeling Approach

To assist understanding, Figure 3.1 depicts the basic notions in the form of a rather informal concept map.

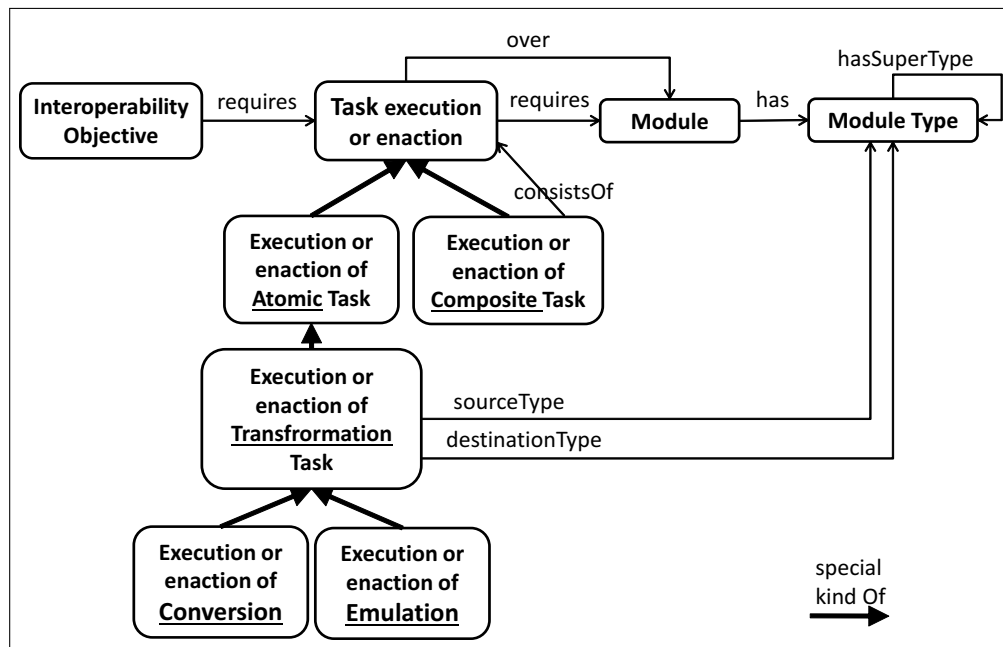


Figure 3.1: Informal concept map

In brief, to achieve performing a task over a module, we need one or more other modules. Each module has a module type, and module types can be hierarchically organized. Now conversion and emulation are special kinds of tasks each having “source” and “destination” module types (broadly speaking).

In section 3.3 we will show how we can model the above using facts (e.g. database tuples) and (Datalog) rules.

3.2 General Methodology

Below we describe six steps of a general methodology for modeling, capturing and managing dependencies for the needs of digital preservation :

1. Identify desired tasks and objectives. This step strongly depends on the nature of the digital objects and the tasks that we want to perform on them. For instance if we suppose our domain is software, we can identify the following tasks: **Edit**, **Compile** and **Run**, while if it is a document we can identify the tasks: **Render**, **Read** and **Edit**.
2. Model the identified tasks and their dependency types. If tasks can be hierarchically organized, then this should be done.
3. Specialize the Rule-based modeling according to the results of the previous step (this will be described in detail in Section 3.4).
4. Capture the dependencies of the digital objects of the archive. This can be done manually, automatically or semi-automatically. Tools like PreScan([9]) can aid this task. In addition this can be done in various levels of granularity: object-level (e.g. for a particular object), type-level (e.g. for all files of type html) and collection-level (e.g. for a collection of images).
5. Customize, use and exploit the dependency services according to the needs. For instance, task-performability services can be articulated with *monitoring* and *notification* services.
6. Evaluate the services in real tasks and curate accordingly the repository (return to Step 1). For instance, suppose that the model fails for one particular module, e.g. the consumer of a package is unable to understand the delivered

module. Such situations indicate that the recorded dependencies are not complete. For example, suppose a user who cannot run a software component, although the computed gap is empty. This can happen if the component has an additional dependency which has not been recorded. A corrective action would be to add this dependency. Analogously, if a user cannot understand a particular research paper this is probably because the paper uses concepts or symbols the user cannot understand. These concepts and symbols are actually dependencies which should be recorded. Synopsizing, empirical testing is a useful guide for defining and enriching the dependency graph.

In the following Sections we describe how we can apply this methodology. Specifically in Section 3.4 we focus on how steps 2 and 3 can capture converters and emulators.

3.3 Modeling Digital Objects, Type Hierarchies, and Profiles

Digital objects, e.g. digital files and their types are represented as facts using predicates that denote their types, e.g. for the three files of our running example, that described in Section 2.4.1, we can have the facts shown in the left column of the following table. Software components are described analogously (e.g. see right column).

Facts	
for digital files	for software components
<code>JavaFile(HelloWorld.java).</code>	<code>TextEditor(vi).</code>
<code>CplusplusFile(HelloWorld.cc).</code>	<code>JVM(jre1.5win)</code>
<code>PascalFile(game.pas).</code>	<code>JVM(jre1.6linux)</code>

Table 3.1: Facts of running examples

Each file can be associated with more than one type. In general we could capture several features of the files (apart from types) using predicates (not necessarily unary), e.g. `LastModifDate(HelloWorld.java, 2013-11-26)`.

The types of the digital files can be organized *hierarchically*, and such taxonomies can be represented with rules, e.g. to define that every `JavaFile` is also a

UTF8File we must add the rule `UTF8File(X) :- JavaFile(X).`

A *profile* is a set of facts, describing the modules available (or assumed to be known) to a user (or community). For example, the profiles of James and Helen are the ticked facts in the corresponding columns of Table 3.1.

3.4 Modeling Task-Dependencies and Task Hierarchies

To implement the steps 2 and 3 of the methodology of Section 3.2, we will also use (IPred) predicates. Specifically, for each real world task we define *two* intensional predicates: one (which is usually unary) to denote the (performability of the) task, and another one (with arity greater than one) for denoting the dependencies of the task. For instance, `Compile>HelloWorld.java)` will denote the compilability of `HelloWorld.java`. Since its compilability depends on the availability of a compiler (specifically a compiler for the Java language), we can express this dependency using a rule of the form: `Compile(X) :- Compilable(X,Y)` where the binary predicate `Compilable(X, Y)` is used for expressing the appropriateness of a `Y` for compiling a `X`. For example, `Compilable>HelloWorld.java, javac 1.6)` expresses that `HelloWorld.java` is compilable by `javac 1.6`. It is beneficial to express such relationships at the class/type level (not at the level of individuals), specifically over the types (and other properties) of the digital objects and software components, i.e. with rules of the form:

```
Compilable(X,Y) :- JavaFile(X), JavaCompiler(Y).
Compilable(X,Y) :- CplusplusFile(X), CplusplusCompiler(Y).
Runnable(X,Y)   :- JavaClassFile(X), JVM(Y).
Editable(X,Y)   :- JavaFile(X), TextEditor(Y).
```

Relations of higher arity can be employed based on the requirements, e.g.:

```
Run(X) :- Runnable(X,Y,Z).
Runnable(X,Y,Z) :- JavaFile(X), Compilable(X,Y), JVM(Z).
```

We can express *hierarchies of tasks* as we did for file type hierarchies, for enabling deductions of the form: “if we can do task A then certainly we can do task B”, e.g. “if we can edit something then certainly we can read it too” expressed as: `Read(X) :- Edit(X)`. Editability here presupposes knowledge of the right symbols set, the one for the intended *information object* as defined in [24].

We can also express *general properties* of task dependencies, like *transitivity*. For example, from `Runnable(a.class, JVM)` and `Runnable(JVM, Windows)` we might want to infer that `Runnable(a.class, Windows)`. Such inferences can be specified by a rule of the form:

```
Runnable(X,Y) :- Runnable(X,Z), Runnable(Z,Y).
```

As another example :

```
IntelligibleBy(X,Y) :- IntelligibleBy(X,Z), IntelligibleBy(Z,Y).
```

This means that if `X` is intelligible by `Z` and `Z` is intelligible by `Y`, then `X` is intelligible by `Y`. This captures the assumptions of the dependency model described in [1] (i.e. the transitivity of dependencies).

3.5 Modeling Converters

Conversions are special kinds of tasks and are modeled differently. In brief to model a converter and a corresponding conversion we have to introduce one unary predicate for modeling the converter (as we did for the types of digital files) and one rule for each conversion that is possible with that converter (specifically one for each supported type-to-type conversion).

In our running example, consider the file `game.pas` (which contains source code in Pascal PL), and the converter `p2c++` from Pascal to C++. Recall that James has a compiler for C++. It follows that James can compile `game.pas` since he can first convert it in C++ (using the converter), then compile it and finally run it. To capture the above scenario it is enough to introduce a predicate for modeling the converters from Pascal to C++, say `ConverterPascal2C++`, and adding the following rule:

```
CplusplusFile(X) :- PascalFile(X), ConverterPascal2Cplusplus(Y).
```

The meaning of this rule is the following: if we have a `PascalFile` `x` and a `ConverterPascal2C` `y` then we can *view* `x` as if it were a `CplusplusFile`.

Since The profile of James will contain the facts `PascalFile(game.pas)` and `ConverterPascal2C++(p2c++)`, we will infer `CplusplusFile(game.pas)`, and subsequently that this file is compilable and runnable.

Finally we should not forget that a converter is itself a module with its own dependencies, and for performing the intended task the converter has to be runnable. Therefore, we have to update the rule as follows:

```
CplusplusFile(X) :- PascalFile(X), ConverterPascal2Cplusplus(Y), Run(Y)
```

3.6 Modeling Emulators

Emulation is again a special kind of task and is modeled differently. Essentially we want to express the following:

If we have :

- (i) a module X which is runnable over Y , and
- (ii) an emulator E of Y over Z (hosting system= Z , target system= Y)

then X is runnable over Z . For example, consider the case where:

- $X=a.exe$ (a file which is executable in Windows operating system),
- $Y=WinOS$ (the Windows operating system),
- $Z=AndroidOS$ (the Android operating system), and
- $E=W4A$ (i.e. an emulator of WinOS over AndroidOS).

In brief, for each available emulator (between a pair of systems) we can introduce a unary predicate for modeling the emulator (as we did for the types of digital files, as well as for the converters), and writing one rule for the emulation.

For example, suppose we have a file named `a.exe` which is executable over WinOS. For this case we would have written:

```
Run(X) :- Runnable(X,Y).
Runnable(X,Y) :- WinExecutable(X), WinOS(Y).
```

and the profile of a user that has this file and runs WinOS would contain the facts `WinExecutable(a.exe)` and `WinOS(mycomputer)`, and by putting them together

it follows that `Run(a.exe)` holds. Now consider a different user who has the file `a.exe` but runs `AndroidOS`. However suppose that he has the emulator `W4A` (i.e. an emulator of `WinOS` over `AndroidOS`). The profile of that user would contain:

```
WinExecutable(a.exe)
AndroidOS(mycomputer) // instead of WinOS(mycomputer)
EmulatorWinAndroid(W4A)
```

To achieve our goal (i.e. to infer that `a.exe` is `runnable`), we have to add one rule for the emulation. We can follow two approaches. The first is to write a rule that concerns the `runnable` predicate, while the second is to write a rule for classifying the system that is equipped with the emulator to the type of the emulated system:

A. Additional rule for Runnable

This relies on adding the following rule:

```
Runnable(X,Y,Z):- WinExecutable(X), EmulatorWinAndroid(Y),
                  AndroidOS(Z).
```

Note that since the profile of the user contains the fact `EmulatorWinAndroid(W4A)` the body of the rule is satisfied (for `X=a.exe`, `Y=W4A`, `Z=myComputer`), i.e. the rule will yield the desired inferred tuple `Runnable(a.exe,W4A,mycomputer)`.

Note that here we added a rule for the `runnable` which has 3 variables signifying the ternary relationship between executable, emulator and hosting environment.

B. Additional type rule (w.r.t. the emulated Behavior)

An alternative modeling approach is to consider that if a system is equipped with one emulator then it can also operate as the emulated system. In our example this can be expressed by the following rule:

```
WinOS(X):- AndroidOS(X), EmulatorWinAndroid(Y).
```

It follows that if the profile of the user has an emulator of type `EmulatorWinAndroid` (here `W4A`) and `mycomputer` is of type `AndroidOS`, then that rule will infer that `WinOS(mycomputer)`, implying that the file `a.exe` will be inferred to be `runnable` due to the basic rule of `runnable` which is independent of emulators i.e. due to the rule:

```
Runnable(X,Y) :- WinExecutable(X), WinOS(Y)
```

Both (A and B) approaches require the introduction of a new unary predicate about the corresponding pair of systems, here `EmulatorWinAndroid`. Approach (A) requires introducing a rule for making the predicate `runnable` “emulator-aware”, while approach (B) requires a rule for classifying the system to the type of the emulated system. Since emulators are modules that can have their own dependencies, they should be runnable in the hosting system. To require their runnability during an emulation we have to update the above rules as follows (notice the last atom in the bodies of the rules):

<pre>A': Runnable(X,Y,Z):- WinExecutable(X), EmulatorWinAndroid(Y), AndroidOS(Z), Runnable(Y,Z)</pre>	<pre> B': WinOS(X):- AndroidOS(X), EmulatorWinAndroid(Y), Runnable(Y,X)</pre>
---	---

3.6.1 Modeling Important Parameters

Sometimes it is important to model the required (important) parameters for the performability of a task. For example, an emulator may need a particular parameter for emulating a particular system. In this case it is beneficial to model this explicitly. Methodologically, it is not suggested to model all parameters, e.g. those of minor importance, but only the crucial ones, those for enabling the required reasoning. For example consider the following rule :

<pre>WinOS(X):- AndroidOS(X), EmulatorWinAndroid(Y), Runnable(Y,X)</pre>
--

and suppose that this emulator needs one particular parameter for emulating windows, say a file `winImg.dat`. One way to capture this, is to extend the above rule as:

<pre>WinOS(X):- AndroidOS(X), EmulatorWinAndroid(Y), Runnable(Y,X), Module(winImg.dat)</pre>
--

where `Module` is the top class of the module type hierarchy. This rule will fire only if the `winImg.dat` is recorded in the system.

3.6.2 Handling Exceptions or Special Cases

Suppose that we know that a given Windows application, can run on Andoid using a Windows emulator, but this is not true for every Windows applications. For example may we know that the emulator W4A cannot run the application `calendar.exe`. To tackle this situation we can add the fact `Exception(calendar.exe,W4A)`, which has a negative interpretation and is read as follows : *the emulator W4A cannot emulate the application calendar.exe*. Now we can extend the approach A as follows:

```
A'' :Runnable(X,Y,Z):-
    WinExecutable(X),
    EmulatorWinAndroid(Y),
    AndroidOS(Z),
    NOT Exception(X,Y),
    Runnable(Y,Z)
```

The difference between the old rule is that we have added a negated `Exception(X,Y)`, meaning that the application X is runnable if the atom `Exception(X,Y)` is false.

Note that the approach B cannot be extended analogously as previous for the approach A, because approach B by default states (as we have seen in Section 3.6) that we can emulate the entire system.

3.7 Modeling Real Converters and Emulators

To evaluate the adequacy of the proposed modeling approach, in this Section we show that some well known converters and emulators can be modeled using our approach.

Texi2HTML converter: Texi2HTML¹ is a Perl script, which converts Texinfo source files to HTML output. Texinfo is the official documentation format of the GNU project.

To model this scenario we must introduce classes for the various module types, i.e. for texi files, for perl scripts, for perl interpreters, and for the particular converter (from texi to HTML). For instance, consider a user who has a `myfile.texi` file, the `strawberry-perl.exe` perl interpreter, and the `Texi2htmlScript.pl` converter (from texi to HTML). The profile of this user will contain the facts:

¹<http://www.nongnu.org/texi2html/>

```

PerlScript(Texi2htmlScript.pl)
PerlInterpreter(strawberry-perl.exe)
TexinfoFile(myfile.texi)
Texi2HTMLConverter(Texi2htmlScript.pl)

```

Note that `Texi2htmlScript.pl` (as any perl script) requires the availability of a Perl interpreter to run, therefore we should add the rule:

```

Runnable(X,Y) :- PerlScript(X), PerlInterpreter(Y)

```

As stated in Chapter 3 we also have to declare a rule for the conversion, in our case the rule:

```

HTML(X) :- TexinfoFile(X), Texi2HTMLConverter(Y), Run(Y)

```

Dioscuri emulator: Dioscuri² is a component-based x86 computer hardware emulator written in Java. Each hardware component is emulated by a software surrogate called a module. By combining several modules the user can configure any computer system, as long as these modules are compatible.

For example consider a user having dioscuri emulator version 0.7.0 (which requires a JVM to run) and suppose he wants to run `Chess.exe`, a 16-bit DOS Application on his computer with the WindowsXp Operating System (`jre1.5win` installed).

Declaring again the appropriate classes, the profile of this user will contain the facts :

```

DOSExecutable(Chess.exe)
WindowsXPOS(mycomputer)
DioscuriEmulator(dioscuri-0.7.0.jar)
JavaByteCode(dioscuri-0.7.0.jar)

```

The execution of a Java ByteCode requires a JVM so:

```

Runnable(X,Y) :- JavaByteCode(X), JVM(Y)

```

From the above now we can write the rule for the emulation:

```

DOSOS(X) :- WindowsXPOS(X), DioscuriEmulator(Y), Runnable(Y,X)

```

²<http://dioscuri.sourceforge.net/>

QEMU emulator: QEMU³ is a generic open source machine emulator and virtualizer that can run an unmodified target operating system. To emulate another machine one needs to have the process emulator (QEMU) and an ISO image of the machine he wants to emulate. For instance, consider a user having the `QEMU1.1` emulator, and an ISO file of the Windows Xp, say `WinXP.iso`, who wants to emulate the WindowsXP OS on his Linux machine. His profile will contain the facts:

```
LinuxOS(mycomputer)
QEMUEmulator(QEMU1.1)
ISOFile(WinXP.iso)
```

Now we can write the rule :

```
WindowsXPOS(X) :- LinuxOS(X), QEMUEmulator(Y), Module(WinXP.iso)
```

As we have stated at Chapter 3, the emulator must be runnable in the hosting system (here `mycomputer`), therefore we have to add a `Runnable` rule, to extend the above rule and reach the following:

```
Runnable(X,Y) :- QEMUEmulator(X), LinuxOS(Y)
WindowsXPOS(X) :- LinuxOS(X), QEMUEmulator(Y)
                  Module(WinXP.iso), Runnable(Y,X)
```

Notice that the user in his profile has the fact `ISOFile(WinXP.iso)`, but the above rule uses the atom `Module(WinXP.iso)`. The rule will fire because `Module` is the top class of the module type hierarchy (i.e. if something belongs to the class `ISOFile` then it also belongs to the class `Module`).

3.8 Synopsis of the Modeling Approach

To synopsise, methodologically for each real world task we define two intensional predicates: one (which is usually unary) to denote the performability of the task, and another one (which is usually binary) for denoting the dependencies of task (e.g. `Read` and `Readable`, `Run` and `Runnable`). To model a *converter* and a corresponding conversion we have to introduce one unary predicate for modeling the converter (as we did for the types of digital files) and one rule for each conversion

³http://wiki.qemu.org/Main_Page

that is possible with that converter (specifically one for each supported type-to-type conversion). To model an *emulator* (between a pair of systems) we introduce a unary predicate for modeling the emulator and writing one rule for the emulation. Regarding the latter we can either write a rule that concerns the **runnable** predicate (approach A), or write a rule for classifying the system that is equipped with the emulator to the type of the emulated system (approach B). Also, and since converters and emulators are themselves modules, they have their own dependencies, and thus their performability and dependencies (actually their runnability) should be modeled too (as in ordinary tasks). Finally, since we should be able to capture special cases (like parameters and exceptions), we have seen how we can model also such cases.

Chapter 4

Reasoning Services

In general, Datalog query answering and methods of logical inference can be exploited for enabling the required inference services (performability, consequences of a hypothetical loss, etc). Here we describe how the reasoning services described in Chapter 2 can be realized using the proposed modeling approach and framework.

4.1 Task-Performability

This service aims at answering if a task can be performed by a user/system. It relies on query answering over the Profiles of the user. E.g. to check if `HelloWorld.cc` is compilable we have to check if `HelloWorld.cc` is in the answer of the query `Compile(X)`.

As we described earlier, *converters* and *emulators* will be taken into account, meaning that a positive answer may be based on a complex sequence of conversions and emulations. This is the essential benefit from the proposed modeling.

For example let us check the performability of the running example, described in section 2.4.1, for the user James. The goal is to check if James can run the `game.pas` file on his mobile phone. Indeed the fact `Runnable(game.pas,smartPhone)` can be derived as shown in the proof tree of Figure 4.1.

In that figure the facts are represented by a rectangle, while the greyed rectangle show the applicable rules. The used facts in this example are:

<pre>PascalFile(game.pas), ConverterPascal2C++(p2c++), WinOS(mycomputer),</pre>

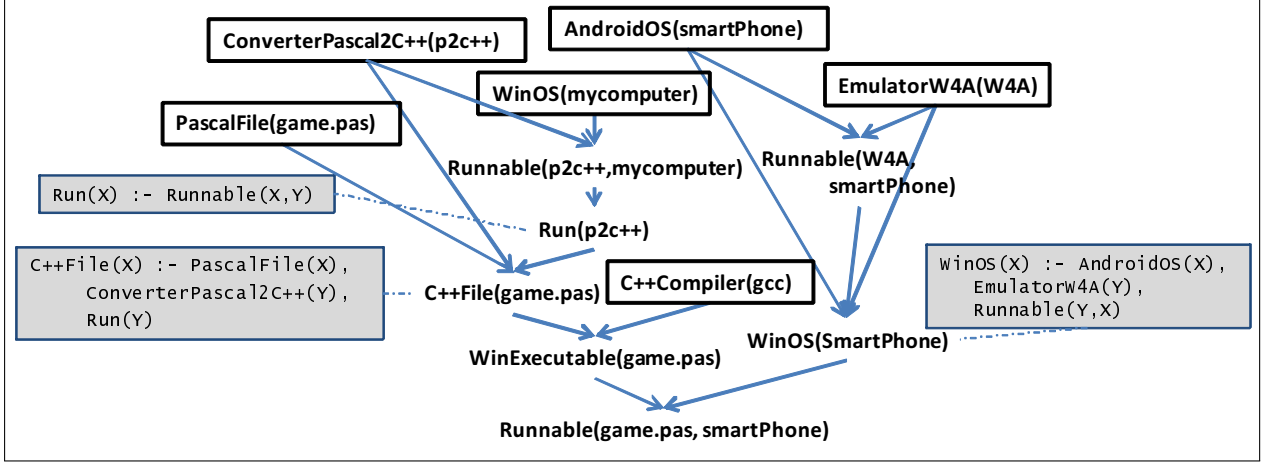


Figure 4.1: The proof tree of the running example

```

AndroidOS(smartPhone),
C++Compiler(gcc),
EmulatorW4A(W4A)

```

Furthermore, classical *automated planning*, e.g. the STRIPS planning method [25], could be applied for returning one of the possible ways to achieve (perform) a task. This is useful in case there are several ways to achieve the task.

4.2 Consequences of a Hypothetical Loss

Suppose that we want to identify the consequences on *editability* after removing a module, say `NotePad`. To do so: (a) we compute the answer of the query `Edit(X)`, let A be the returned set of elements, (b) we delete `NotePad` from the database and we do the same, let B be the returned set of elements¹, and (c) we compute and return the elements in $A \setminus B$ (they are the ones that will be affected).

4.3 Computation of Gaps (Missing Modules)

The gap is actually the set of facts that are missing and are needed to perform a task. There can be more than one way to fill a gap due to the disjunctive nature

¹ In an implementation over Prolog, we could use the *retract* feature to delete a fact from the database.

of dependencies since the same predicate can be the head of more than one rules (e.g. the predicate `TextEditor` in the example of Section 3.6). One method to fill the gaps is to construct and visualize an AND-OR graph that contains information about only the related facts and rules. Such an approach is described in [26]. An alternative (or complementary) approach is to allow the user to *gradually* explore the possibilities and navigate through the possible paths. The implemented system that is described in Chapter 6 follows this approach.

Chapter 5

Implementation

5.1 Possible Implementation Technologies

There are several possible implementation approaches. Below we describe them in brief:

Prolog is a declarative logic programming language, where a program is a set of Horn clauses describing the data and the relations between them. The proposed approach can be straightforwardly expressed in Prolog. There are several approaches that extend the Prolog and can be used for the implementation. For instance *XSB*¹ extends Prolog with tabled resolution and HiLog (a standard extension of Prolog permitting limited higher-order logic programming). Tabled resolution is useful for recursive query computation, allowing programs to terminate correctly in many cases where Prolog does not.

A **Semantic Web** approach can be used. The SWRL (Semantic Web Rule Language) [27] is a combination of OWL DL and OWL Lite [28] with the Unary/Binary Datalog RuleML². SWRL provides the ability to write Horn-like rules expressed in terms of OWL concepts to infer new knowledge from existing OWL KB. For instance, each type predicate can be expressed as a class. Each profile can be expressed as an OWL class whose instances are the modules available to that profile (we exploit the multiple classification of SW languages). Module type hierarchies can be expressed through *subclassOf* relationships between the corresponding classes. All rules regarding performability and the hierarchical organization of

¹<http://xsb.sourceforge.net/>

²<http://ruleml.org>

tasks can be expressed as SWRL rules. A limitation of the SWRL approach is that ternary or higher rules cannot be captured by the rules. An alternative Semantic Web approach, on which we focus on this thesis, is to use triple-stores and exploit its query-based inference capabilities (more in Chapter 6).

In a **DBMS**-approach all facts can be stored in a relational database, while *Recursive SQL* can be used for expressing the rules. Specifically, each type predicate can be expressed as a relational table with tuples the modules of that type. Each profile can be expressed as an additional relational table, whose tuples will be the modules known by that profile. All rules regarding task performability, hierarchical organosis of tasks, and the module type hierarchies, can be expressed as datalog queries. Note that there are many commercial SQL servers that support the SQL:1999 syntax regarding recursive SQL (e.g. Microsoft SQL Server 2005, Oracle 9i, IBM DB2).

Just indicatively, Table 5.1 synthesizes the various implementation approaches.

What	DB-approach	Semantic Web-approach
ModuleType predicates	relational table	class
Facts regarding Module (and their types)	tuples	class instances
DC Profile	relational table	class
DC Profiles Contents	tuples	class instances
Task predicates	IDB predicates	predicates appearing in rules
Task Type Hierarchy	datalog rules, or isa if an ORDBMS is used	<i>subclassOf</i>
Performability	datalog queries (recursive SQL)	rules

Table 5.1: Implementation Approaches

5.2 On RDF/S-based Implementations

Here we describe one Semantic Web-based implementation using RDF/S and *Open-Link Virtuoso* which is a general purpose RDF triple store with extensive SPARQL and RDF support [29]. Its internal storage method is relational, i.e. RDF triples are stored in tables in the form of quads (g, s, p, o) where g represents the graph,

s the subject, p the predicate and o the object. We decided to use this system because of its inference capabilities, namely *backward chaining* reasoning, meaning that it does not materialize all inferred facts, but computes them at query level. Its reasoner covers the related entailment rules of `rdfs:subClassOf` and `rdfs:subPropertyOf`, while *user defined custom inference rules* can be expressed using *rule sets*. Practically this means that transitive relations (i.e. *subClassof*, *subPropertyOf*, etc.) are not physically stored in the knowledge base, but they are added to the result set at query answering. *Transitivity* is also supported in two different ways. Given a RDF schema and a rule associated with that schema, the predicates `rdfs:subClassOf` and `rdfs:subPropertyOf` are recognized and the inferred triples are derived when needed. In case of another predicate, the option for transitivity has to be declared in the query.

For our case, we have to “translate” our facts and rules to quads of the form (g, s, p, o) which are actually RDF triples contained in a graph g . The support of different graphs is very useful for the cases of profiles; we can use a different graph for each profile. We will start by showing how facts can be “translated” to RDF quads and later we will show how inference rules can be expressed using ASK and CONSTRUCT or INSERT SPARQL queries. For better readability of the SPARQL statements below we omit namespace declarations.

5.2.1 Modules

Module types are modeled using RDF classes while the actual modules are instances of these classes. Module type hierarchies can be defined using the `rdfs:subClassOf` relationship. For example the fact `JavaFile(HelloWorld.java)` and the rule for defining the module type hierarchy `TextFile(X) :- JavaFile(X)` will be expressed using the following quads:

```
g, <JavaFile>, rdf:type, rdfs:Class
g, <TextFile>, rdf:type, rdfs:Class
g, <JavaFile>, rdfs:subClassOf, <TextFile>
g, <HelloWorld.java>, rdf:type, <JavaFile>
```

5.2.2 Profiles

We exploit the availability of graphs to model different profiles, e.g. we can model the profiles of James and Helen (including only some indicative modules), as follows:

```
<jGrph>, <NotePad>, rdf:type, <TextEditor>
<jGrph>, <HelloWorld.java>, rdf:type, <JavaFile>
<jGrph>, <javac_1_6>, rdf:type, <JavaCompiler>
<hGrph>, <VI>, rdf:type, <TextEditor>
<hGrph>, <jre_1_5>, rdf:type, <JavaVirtualMachine>
```

5.2.3 Dependencies

The rules regarding the performability of tasks and their dependencies are transformed to appropriate SPARQL CONSTRUCT statements which produce the required inferred triples. For example, the rule about the compilability of Java files ($\text{Compilable}(X,Y) \text{ :- JavaFile}(X), \text{JavaCompiler}(Y)$) is expressed as:

```
CONSTRUCT{?x <compilable> ?y}
WHERE{?x rdf:type <JavaFile>.
      ?y rdf:type <JavaCompiler>}
```

To capture the compilability of other kinds of source files (i.e. C++, pascal etc.) we extend the previous statement using the UNION keyword (this is in accordance with the Datalog-based rules; multiple rules with the same head have union semantics). For example the case of Java and C++ is captured by:

```
CONSTRUCT{?x <compilable> ?y}
WHERE{
  {?x rdf:type <JavaFile>.
    ?y rdf:type <JavaCompiler>}
  UNION
  {?x rdf:type <C++File>.
    ?y rdf:type <C++Compiler>}
}
```

Finally the unary predicate for the performability of task, here `Compile`, is expressed as:


```
CONSTRUCT{?x rdf:type <Compile>}
WHERE{ {?x <compilable> ?y} }
```

5.2.4 Converters

The rules regarding conversion are modeled analogously, e.g. for the case of a converter from Pascal to C++ we produce:

```
CONSTRUCT{?x rdf:type <CplusplusFile>}
WHERE{?x rdf:type <PascalFile>.
      ?y rdf:type <ConverterPascal2Cplusplus>.
      ?y rdf:type <Runnable>}
```

Note the last condition refers in an inferred type triple (Runnable). If there are more than one converters that change modules to a specific module type then the construct statement is extended using several WHERE clauses separated by UNIONS, as shown previously.

5.2.5 Emulators

Consider the scenario described in Chapter 3, i.e. a user wanting to run `a.exe` upon his Android operating system. The approach B (which does not require expressing any predicate with three variables), can be expressed by:

```
CONSTRUCT{?x rdf:type <WindowsOS>}
WHERE{?x rdf:type <AndroidOS>.
      ?y rdf:type <EmulatorWin4Android>.
      ?y <runnable> ?x}
```

If the emulator needs a particular parameter, as for example the module `winImg.dat` which we have described in Chapter 3 we have to add an extra triple on the previous query for this module, so we model the emulator as :

```
CONSTRUCT{?x rdf:type <WindowsOS>}
WHERE{?x rdf:type <AndroidOS>.
      ?y rdf:type <EmulatorWin4Android>.
      ?y <runnable> ?x.
      <winImg.dat> rdf:type <Module>}
```

5.2.6 Services

To realize the reasoning services (e.g. task performability, consequences of a hypothetical loss, etc), we rely on SPARQL queries. For example to answer if the file `HelloWorld.java` can be compiled we can send the INSERT query about the compilability of the files (as shown previously) and then perform the following ASK query on the entailed triples:

```
ASK{<HelloWorld.java> <compilable> ?y}
```

If this query returns true then there is at least one appropriate module for compiling the file.

The “Consequences of a Hypothetical Loss service” requires SELECT and DELETE SPARQL queries (as discussed in Chapter 4). For example to find those modules whose *editability* will be affected if we remove the module `Notepad`, we have to perform :

```
SELECT ?x
WHERE {?x rdf:type <Edit>}

DELETE <Notepad> rdf:type <TextEditor>
```

From the select query we get a set A containing all modules which are editable. Then we remove the triple about `Notepad` and perform again the select query, getting a new set B . The set difference $A \setminus B$ will reveal the modules that will be affected. If empty this means that there will be no risk in deleting the `Notepad`.

5.3 Evolution, Representation of Profiles and Limitations

5.3.1 Inference and Evolution

The explicit triples can be stored in one graphspace, say G_e , and an additional one, say G_c , can store the explicit plus the inferred triples as produced by the INSERT statements. All queries (and reasoning services) should be based on G_c . The advantage of this approach is that query answering is fast. The downside is that this policy increases the triples that are stored (in G_c) and makes *updating*

more cumbersome. For instance, if we use G_e for updates, then after such an update we have to reconstruct from scratch G_c .

It follows from the above that using two different graphs (G_c and G_e), it is not so efficient if the dataset is big. An alternative approach would be to use *views* of the semantic store. In general, a *view* is a virtual part of the base store. Using a view we do not have to create the G_c for the inferred triples, instead we create a view over G_e that contains all rules. There are some works (e.g. [30], [31], [32]) that propose view mechanisms for Semantic Web data, but most of them are in a preliminary phase. The work presented in [31], which describes a view language based on RQL [33] query language (a query language that takes semantics of RDFS ontologies into account), could satisfy our requirements. They make a distinction between views on properties and views on classes. Views on properties or on classes can be defined using arbitrary queries. Using this approach we can use queries to create new properties and classes views, instead of issuing CONSTRUCT queries in a new graph space. There is an implementation of this work, that implements RDFS Semantics, involving the computation of transitive closure of the class and property hierarchies. However, the current standard is SPARQL (not RQL), and the aforementioned implementation is no longer available.

Another work that could be used in our approach is [34]. They propose an algorithm (SQR) for rewriting a query of a *virtual view* to a equivalent query over the underlying data (in case of an RDF database) avoiding the cost entailed by view materialization. The algorithm takes as input the views (written in SPARQL) and a query Q on views, and returns a rewriting Q' as a union of conjunctive queries. The Q' now is over the vocabulary of the underlying data. Finally they perform some optimizations on the algorithm to reduce the cost.

Since there is not a mature and expressive view language for SPARQL, in our prototype (described in Chapter 6) we adopt the two-graphs approach that we described in the beginning.

5.3.2 Representation of Profiles

An issue is how the *profiles* should be stored. As we have defined in section 2.2 a user can have a profile which is actually the set of modules that are assumed to be known. A system should be able to manage more than one profile, and the contents

of a profile determine the rules that can be fired. In an RDF implementation a way to represent different profiles is to use separate *graph spaces* for each one and this is the solution that we use for our implementation.

The advantage is that is a fast and a “clear” solution, but we have to use extra space for the graphs. In addition, in a policy where the inferred (by the rules) triples are stored, the graph space of each profile should keep also the corresponding inferred triples.

5.3.3 Limitations of RDF/S

One limitation of the RDF/S representation framework is that ternary or higher order relationships cannot be captured by triples. In this case the conceptual modeling has to break such relationships to a set of binary. For example, a relationship (a, b, c, d) can be represented by (a, x) , (x, b) , (x, c) , (x, d) where x is an auxiliary node. *RDF blank nodes* can be used for this purpose, i.e. for representing x without having to assign a name (identity) to that node. To be more specific, consider the ternary relation that is produced by the following rule (that was described in Section 3.6):

```
Runnable(X,Y,Z) :- WinExecutable(X),
                    EmulatorWinAndroid(Y),
                    AndroidOS(Z)
```

Also assume the following assignment $X=a.exe$, $Y=W4A$, $Z=mycomputer$. To express on RDF (with the aid of blank nodes), the outcome of the above rule, we should construct the following triples:

```
{<a.exe> <runnable> _:bn}.
{_:bn <runnable> <W4A>}.
{_:bn <runnable> <mycomputer>}.

```

To achieve this we can use a variation of the queries that we have seen on Chapter 5. Specifically to construct the triples of the above example we can use the following query :

```
CONSTRUCT{?x <runnable> _:bn.
          _:bn <runnable> ?y.
          _:bn <runnable> ?z}
```

```

WHERE{
  ?x rdf:type <WinExecutable>.
  ?y rdf:type <EmulatorW4A>.
  ?z rdf:type <AndroidOS>
}

```

5.4 Proof-of-Concept Dataset and Repository

The objective of this dataset is to allow checking the correctness of our RDF/S implementation method, and for this reason it contains all examples that are described in this thesis.

For instance, we have created and loaded a N-triple file that contains the triples that define the schema (classes and properties) and the facts for the described example in Section 2.4.1. All explicit triples are entered in one graph space, say *j_graph* (James' graph). We adopt an additional graph space, say *j_graph_compl* ("compl" from completed) that stores all explicit plus all inferred triples. The inferred triples are produced by the INSERT statements that correspond to the rules. All queries (and reasoning services) are based on *j_graph_compl*.

Moreover, all facts and rules of the examples of our approach (including the examples of real converters and emulators which are described in Section 3.7) have been stored in a prototype repository, accessible through a SPARQL endpoint <http://62.217.127.222:8890/sparql>. Specifically the graph *j_graph_compl* contains all the produced triples from the aforementioned examples. Any user or service can connect for executing the desired SPARQL queries. We used it for validating that the implementation behaves as specified by the theory.

In addition, we have set up a system for exploring the contents of the KB. We have use the OpenRDF Sesame³ system. The system is publicly accessible⁴. Using the left tab, on the explore area, the user can see the contents of the KB and also can formulate SPARQL queries. Figure 5.1 shows an indicative screenshot of the system.

³<http://www.openrdf.org/index.jsp>

⁴<http://139.91.183.63:8080/openrdf-workbench/repositories/VirtuosoRep/contexts>

Workbench OpenRDF

Sesame server

Repositories
 New repository
 Delete repository

Explore
 Summary
 Namespaces
 Contexts
 Types
 Explore
 Query
 Export

Modify
 SPARQL Update
 Add
 Remove
 Clear

System
 Information

Current Selections:
 Sesame server: <http://139.91.183.63:8080/openrdf-sesame> [\[change\]](#)
 Repository: [Virtuoso Sesame Repository \(VirtuosoRep\)](#) [\[change\]](#)

Explore (<http://ipres/j_graph_compl>)

The results shown maybe truncated.

Super Classes

- <http://data.ipres.gr/TextFile>

Sub Classes

- <http://data.ipres.gr/JavaFile>

Subject	Predicate	Object	Context
http://data.ipres.gr/JVM	<code>rdfs:type</code>	<code>rdfs:Class</code>	http://ipres/i_graph_compl
http://data.ipres.gr/JVM	<code>rdfs:type</code>	http://data.ipres.gr/Run	http://ipres/i_graph_compl
http://data.ipres.gr/TextEditor	<code>rdfs:type</code>	<code>rdfs:Class</code>	http://ipres/i_graph_compl
http://data.ipres.gr/ire1.5win	<code>rdfs:type</code>	http://data.ipres.gr/JVM	http://ipres/i_graph_compl
http://data.ipres.gr/ire1.5win	http://data.ipres.gr/runnable	http://data.ipres.gr/WindowsOS	http://ipres/i_graph_compl
http://data.ipres.gr/ire1.5win	http://data.ipres.gr/runnable	http://data.ipres.gr/mvcomputer	http://ipres/i_graph_compl
http://data.ipres.gr/vi	<code>rdfs:type</code>	http://data.ipres.gr/TextEditor	http://ipres/i_graph_compl
http://data.ipres.gr/TextFile	<code>rdfs:type</code>	<code>rdfs:Class</code>	http://ipres/i_graph_compl
http://data.ipres.gr/C++File	<code>rdfs:type</code>	<code>rdfs:Class</code>	http://ipres/i_graph_compl
http://data.ipres.gr/PascalFile	<code>rdfs:type</code>	<code>rdfs:Class</code>	http://ipres/i_graph_compl
http://data.ipres.gr/C++Compiler	<code>rdfs:type</code>	<code>rdfs:Class</code>	http://ipres/i_graph_compl
http://data.ipres.gr/JavaCompiler	<code>rdfs:type</code>	<code>rdfs:Class</code>	http://ipres/i_graph_compl
http://data.ipres.gr/ConverterP2C++	<code>rdfs:type</code>	<code>rdfs:Class</code>	http://ipres/i_graph_compl
http://data.ipres.gr/EmulatorW4A	<code>rdfs:type</code>	<code>rdfs:Class</code>	http://ipres/i_graph_compl
http://data.ipres.gr/WinOS	<code>rdfs:type</code>	<code>rdfs:Class</code>	http://ipres/i_graph_compl
http://data.ipres.gr/AndroidOS	<code>rdfs:type</code>	<code>rdfs:Class</code>	http://ipres/i_graph_compl
http://data.ipres.gr/WinExecutable	<code>rdfs:type</code>	<code>rdfs:Class</code>	http://ipres/i_graph_compl
http://data.ipres.gr/Compile	<code>rdfs:type</code>	<code>rdfs:Class</code>	http://ipres/i_graph_compl
http://data.ipres.gr/WindowsOS	<code>rdfs:type</code>	<code>rdfs:Class</code>	http://ipres/i_graph_compl
http://data.ipres.gr/Edit	<code>rdfs:type</code>	<code>rdfs:Class</code>	http://ipres/i_graph_compl

Figure 5.1: Exploration System

Chapter 6

Epimenides: A Proof-of-Concept System

For proving the technical feasibility, as well as for demonstration and dissemination purposes, we have build a web accessible system, called **Epimenides**¹.

6.1 Use Cases

The Use Case diagram that provides an overview of the supported functionality is given in Figure 6.1. The application can be used by several users, and each can build and maintain his/her own profile. To be flexible, a gradual method for the definition of profiles is supported. The Knowledge Base (KB) of this system currently contains 2,225 RDF triples and it is described in Section 6.4, while the main scenario is described next.

After login, *the user can upload a digital object* (file or zipped files) and select the task whose performability he wants to check. The system then checks the dependencies and computes the corresponding gap. The curator can define new tasks to the system. To identify the dependencies of the uploaded objects, the system exploits the extension of the object (like .pdf, .doc, .docx), and its KB already stores the dependencies of some widely used file types. The identified dependencies are then shown to the user. The user can then *add* those that (s)he already has, and this is actually the method for defining his profile *gradually*. In

¹<http://www.ics.forth.gr/isl/epimenides/>

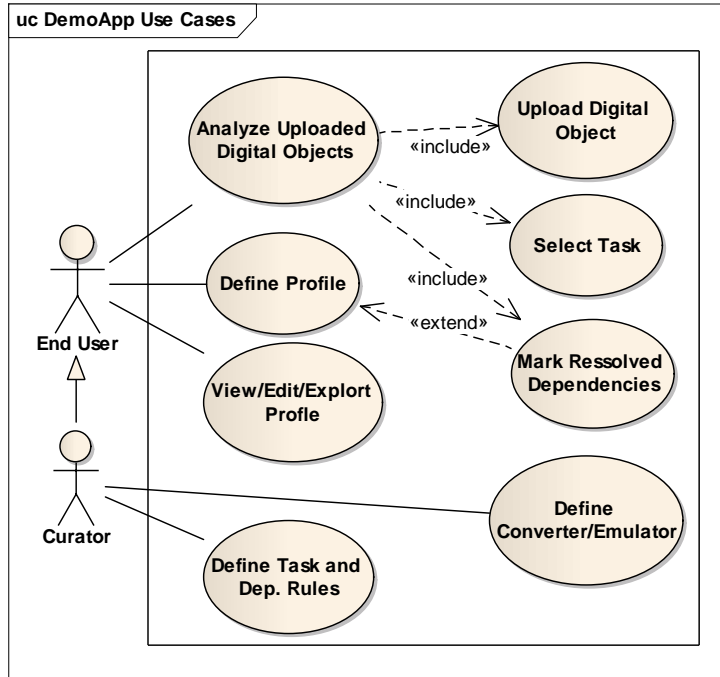


Figure 6.1: Use Case Diagram of Epimenides

this way he does not have to define his profile in one shot. The system stores the profiles of each user (those modules marked as “I have them”) to the RDF storage. The profiles are stored according to the method we have described in Section 5.3.2, using different graph spaces for each user/profile.

6.2 Deployment of Epimenides

The architecture of **Epimenides** is based on the MVC (Model View Controller) pattern, meaning that all business logic is implemented in Java Servlets and all communication and data transfer issues are dealt with the use of Java Beans. The presentation of data is specified using JSP pages in order to separate the presentation design from the application logic, making easier the extension and modification of the system.

Epimenides is using the Apache Tomcat² 7.0.3 web server while as a triple store uses the OpenLink Virtuoso³ 06.01.3127 version. The Virtuoso Jena RDF

²<http://tomcat.apache.org/>

³<http://virtuoso.openlinksw.com/>

Data Provider⁴ is used for the communication with the triplestore. Figure 6.2 shows the deployment diagram of Epimenides.

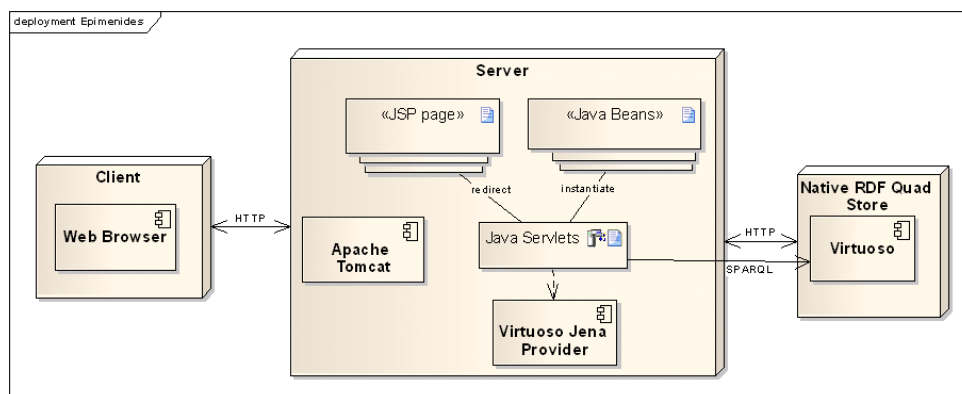


Figure 6.2: The deployment diagram of Epimenides

6.3 User Interface

The user interface contains a menu divided in three sections as shown in Figure 6.3. The first contains the main option of the application: “Upload Digital Object”. The “Manage Profile” section contains options available to any user (simple users). The user can add/delete modules to his profile.

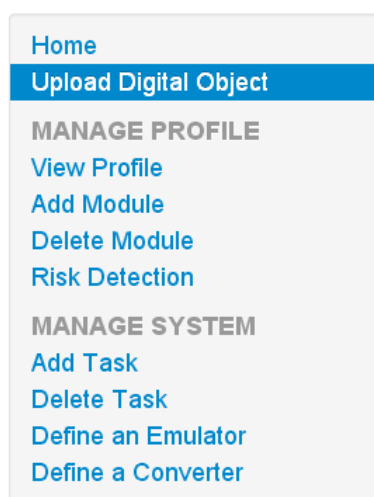


Figure 6.3: Main functionality of Epimenides

⁴<http://www.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtJenaProvider>

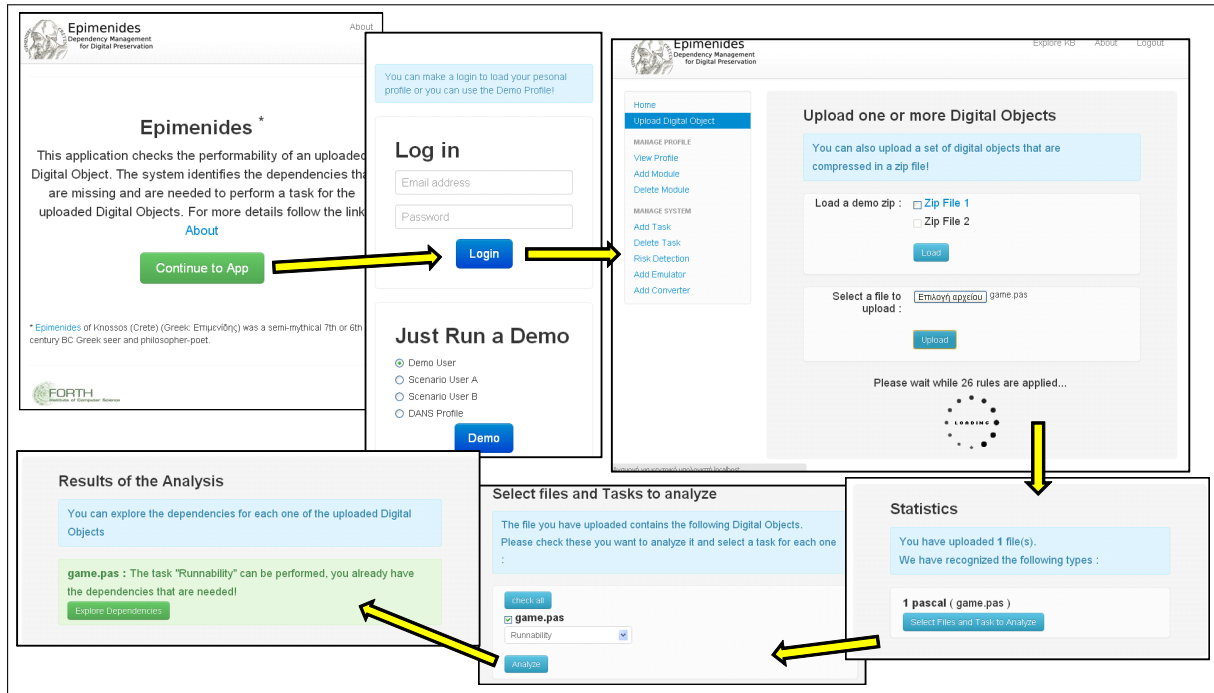


Figure 6.4: Checking the performativity over a digital object

The “Manage System” section contains options for a curator user. Such a user has also the ability to define Tasks, Emulators and Converters. To properly add a Task, an Emulator or a Converter one has to provide extra information from which the application will produce the required rules (as we will describe in sections 6.5 and 6.6). A simple user can add to his profile an emulator X, only if it has been properly defined from a curator user (and consequently the application has produced the required rules).

Figure 6.4 summarizes the interaction between a user and the Epimenides for checking the performativity of a task over a digital object. More analytical screenshots of the application are provided in Section 6.7.

6.4 The Knowledge Base of Epimenides

Figure 6.5 shows the architecture of the system’s KB. As we have mentioned before, the profile of a user contains triples with the modules that holds on his system, while the application contains information about the dependencies that are needed to execute a task. For the representation of the modules the KB contains all the

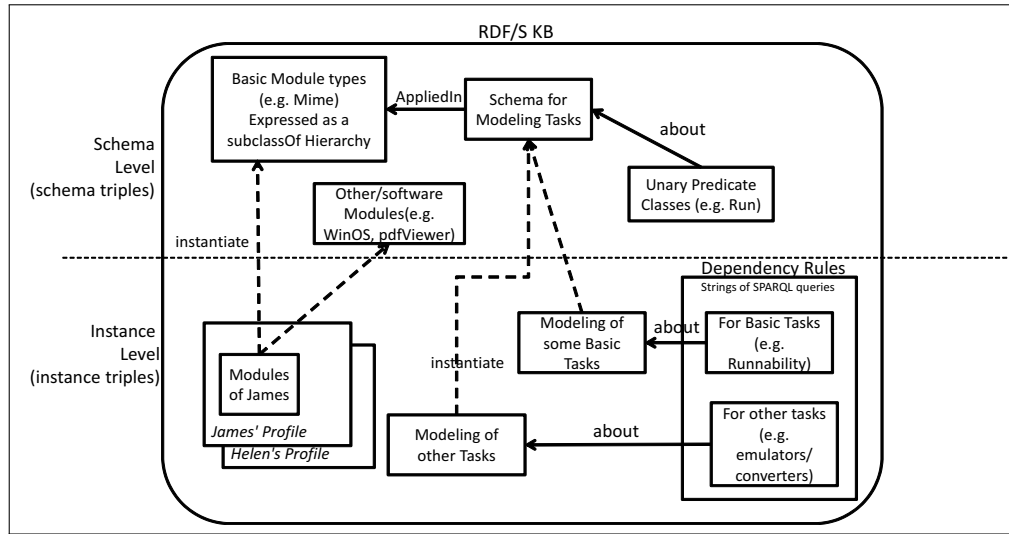


Figure 6.5: Architecture of the KB

MIME media types⁵ expressed as a `subClassOf` hierarchy (this hierarchy is shown in the left of Figure 6.6). The dependency rules are also stored in the KB as strings of SPARQL queries. Finally the KB also contains information about tasks.

To explain the structure of the KB we shall use an example that is illustrated in Figure 6.6. Suppose a user that his profile contains only the module *WinOS* and he uses the application for first time. The user uploads a file, say *f*, and the system by its filetype extension (suppose `.exe`), or by analyzing the contents (e.g. by using tools like Jhove⁶ or JMimeMagic library⁷), can realize that the uploaded file is an executable file, and that belongs to the `"application/octet-stream"` MIME type and consequently to the `octet-stream` class of the KB (as shown in Figure 6.6). To achieve this for the first case (using the file extension) any MIME type class in the KB has the property `hasExtension`. In this way the KB contains the triple (`<octet-stream>`, `<hasExtension>`, `".exe"`). This means that the system from the extension understands the class that models the file type by running the SPARQL query :

```
SELECT ?className
WHERE{?className <hasExtension> ".exe"}
```

⁵Multipurpose Internet Mail Extensions (MIME) is an Internet standard that extends the format of email.

⁶<http://jhove.sourceforge.net/>

⁷<http://jmimemagic.sourceforge.net/index.html>



By knowing the class that models the type of f , the system can find the tasks that usually make sense to apply to the uploaded file by the property `appliedIn` of the KB, which has domain a task and range a MIME type. The system shows a list of all these tasks, returned by the following query :

The user can select one of the retrieved tasks and the next step for the system is to check if this task can be performed. This can be done by the task-performability service (as we have described in Chapter 4). To implement this service we have added in the class **Task** of the KB, the following predicates (also shown in Figure 6.6):

- `hasName` (literal) - the name of the task e.g. *Runnability*
- `hasPerformabilityName` (literal) - the unary predicate that denotes the per-

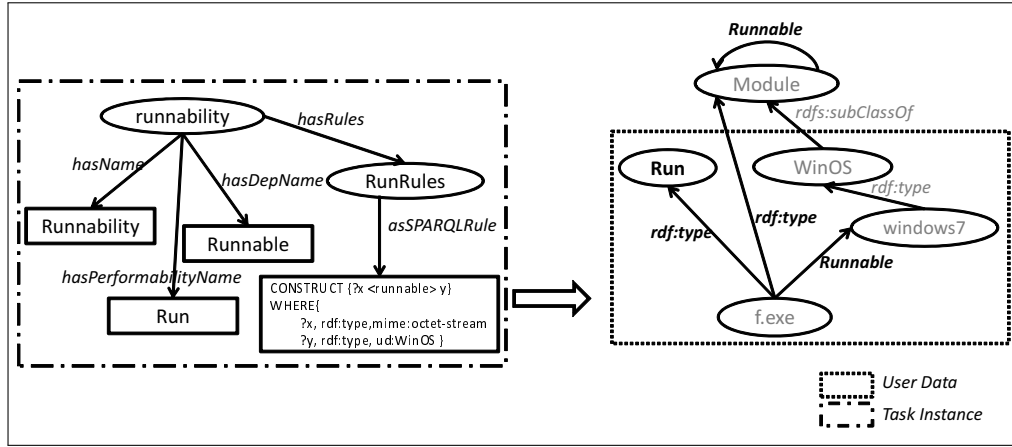


Figure 6.7: Operational KB

formability of the task e.g. *Run*

- **hasDepName** (literal) - the binary predicate that denotes the dependencies of task e.g. *runnable*
- **hasRules** (Rules) - the rules that determine when the task can be performed.

To perform the performability checking service, the system produces the *Operational KB* in which new classes and new properties are created and populated. The name of new classes and properties are determined by the properties of KB **hasPerformabilityName** and **hasDepName** respectively. The *Operational KB*, for short *OKB*, is a superset of KB. Specifically it contains the results of the application of *all* rules that KB contains. For the creation of the OKB we retrieve all the rules and we check if they can be applied (using the contents of the KB). Whenever a rule is applied the OKB is updated (the corresponding classes and properties are created), and we start to check again for rules that can be applied in the new OKB. Essentially we apply a *fixpoint* method of the Datalog language. In this way, query answering can indeed support the desired services for task performability, taking also into account the emulators and the converters.

In our case suppose that the user has selected the task with name “*Runnability*”. All the rules of the KB are applied and the OKB is created. In Figure 6.7 you can see the OKB for our example, its right side shows how the *User Data* are changed when the OKB is produced (with bold are represented the new produced resources).

The class **Run** and the property **Runnable** have already been created in the

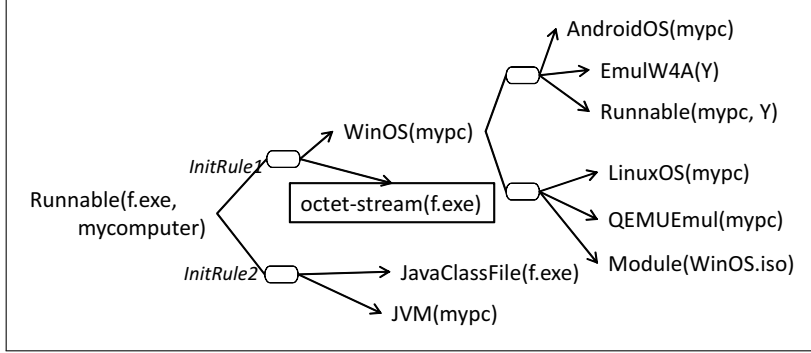


Figure 6.8: Gradual Expansion

OKB. Now the system using the property `hasPerformabilityName`, issues the query `Run(f)` in the profile of the user (in OKB). Obviously, the answer of the query in our example is true, as you can see in Figure 6.7, therefore the system informs the user that this task can be finally performed. In case the selected task could not be performed, the system should inform the user for the missing dependencies.

To determine the dependencies that are missing and are required for performing the selected task (the Computation of Gaps service as described in Chapter 4), the system uses the Dependency Rules that are stored in the KB. Specifically at first it retrieves from the property `asSPARQLRule` of each one rule of the selected task, the direct dependencies, which actually form a set of atoms. These atoms are shown to the user and he can ask the system to show how an atom can be satisfied. In this case the system explores the KB for rules (including rules for emulators/converters) that has as head the selected atom. The above procedure is repeated for the new rules. In this way a gradual expansion is created as the user gradually explores the possible paths.

Figure 6.8 shows an example of the above procedure. This example corresponds to the case where the user cannot perform the `Runnability` task for the uploaded file `f.exe`. The system retrieves and shows to the user the direct dependencies as specified by two rules denoted by `InitRule1` and `InitRule2`. This means that to turn `f.exe` runnable either `InitRule1` OR `InitRule2` should be satisfied. The body atoms of each rule are shown, e.g. `InitRule1` requires a `WinOS` AND an `octet-stream` file. Subsequently the user can request from the system to show how each of these atoms, say `WinOS(mycomputer)` of `InitRule1`, can be satisfied and/or added to his profile. Analogously, the system retrieves and shows the rules

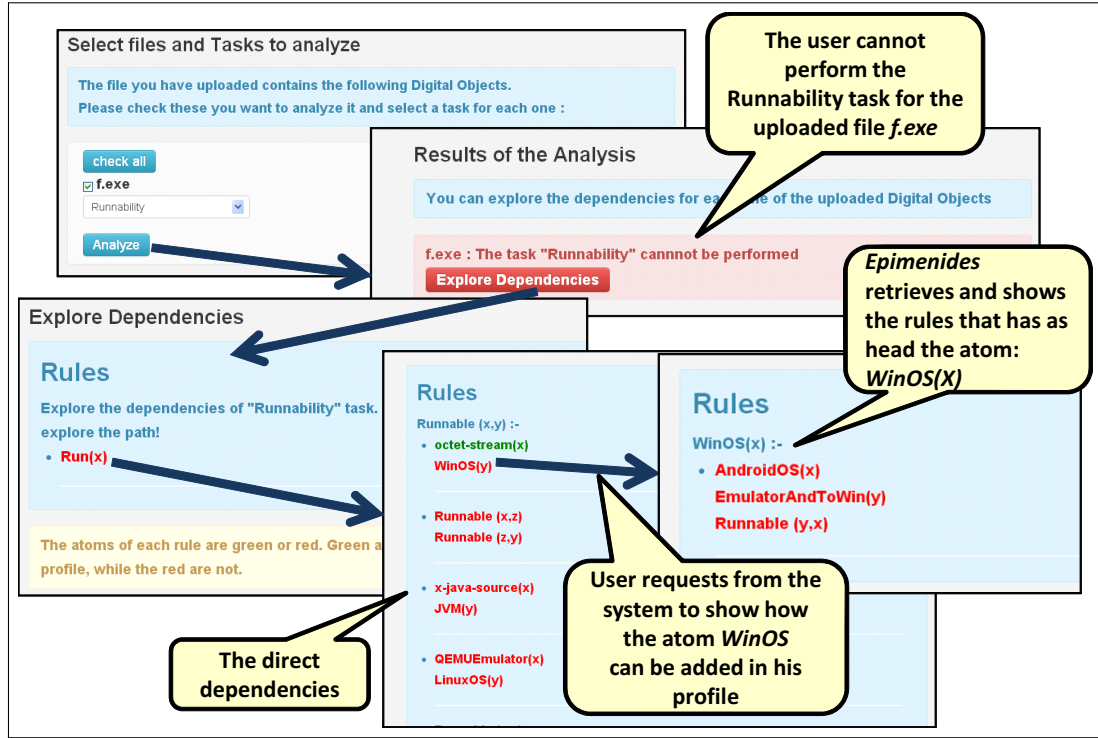


Figure 6.9: The Gradual Expansion in Epimenides

that have as head the atom *WinOS(X)*.

This process is supported also by *Epimenides*, and Figure 6.9 shows a series of screenshots that illustrate it. We should also mention that during the above process the system informs the user about those atoms which are already satisfied by his profile. This assists the user to decide which paths are useful to explore. In Figure 6.8 these atoms are enclosed in rectangle, while *Epimenides* colors them green (see Figure 6.9).

6.4.1 The Current Knowledge Base

Currently the Knowledge Base that we have created for a demo user contains 657 Module Types, including 647 MIME Type Modules. It also contains information about three tasks (readability, runnability, rendering). Furthermore, for each of the 647 Mime Types the knowledge base contains extra information (e.g. the extension of a mime type). As regards the dependencies of the tasks, 53 rules have been specified. In total, the knowledge base contains 2,304 RDF triples. Note, as we have already mentioned, that a user can enrich the knowledge base by adding

his/her own Module Types, Tasks and Rules using the prototype system. The RDF Schema of the KB is given in the Appendix A.1.

We should clarify that the KB composed from two different graph spaces: the user profile graph space that contains information for the modules that a user holds, and the system's graph space that contains information about the rules, the tasks and the modules.

Also the current version of the system inserts to the RDF storage the inferred triples, as we have described in section 5.3.1, and does not use a view language.

6.5 Aiding the Ingestion of Tasks

Above we have described the use cases for end users. The job of the curator could also be assisted by providing a simple method for adding tasks and modeling the corresponding dependencies. This is related to the use case named “Define Task and Dep. Rules” in Figure 6.1. The curator can enrich the system to support extra tasks and rules. He should provide some input and the application produces the required rules. Specifically the curator should provide :

- The unary predicate that denotes the performability of the task (e.g. Edit),
- the MIME type/s that can be applied in this task (e.g. text/plain),
- the module/s that is/are required for the selected MIME Type (e.g. Text Editor),
- and optionally the task which the performability can be implemented by the new task (e.g. Readability).

6.6 Aiding the Ingestion of Converters and Emulators

Again, and for aiding the job of the curator in defining new emulators here we describe an automatic method where the user through a user interface provides some input, and the tool outputs the required facts and rules. This is related to the use case named “Define Converter/Emulator” in Figure 6.1. The method is based in Chapter 3 and on the examples for the real dataset in Section 3.7.

In brief, to represent an emulator the user has to provide:

- What this emulator emulates (e.g. the WindowsXp Operating System), let denote this input by A .

- In what System the emulator runs (e.g. Linux Operating System), let denote this input by B .
- The URI (or just filepath) and the name of the emulator that we want to register, let denote this by $Epath$ and E respectively.
- Files or other modules that the emulator uses (e.g. ISOFile). Let denote such input by P_1, \dots, P_k .

Let A_{pred} denote the predicate corresponding to A , e.g. the URI of the RDF class WindowsXP (in the context of an interactive system the user could search and select this from the list of registered modules).

Let B_{pred} denote the predicate corresponding to B , e.g. the URI of the RDF class LinuxOS. (again in the context of an interactive system the user could search and select this from the list of registered modules).

Regarding P_1, \dots, P_k , and assuming that each one of these is specified by a URI, they will be used for extending the emulator rules (as described earlier).

Now we describe exactly what facts and rules should be produced by the above input. At first the system creates a new RDF class E and also creates the fact $E(Epath)$. Then it creates the following rule:

```
A_pred(X) :- B_pred(X), E_pred(Y), Runnable(Y,X),
             Module(P_1), ..., Module(P_k).
```

For the runnability of the emulator, we define that the emulator is runnable on the System that the emulator runs, so the system creates the rule :

```
Runnable(X,Y) :- E_pred(X), B_pred(Y)
```

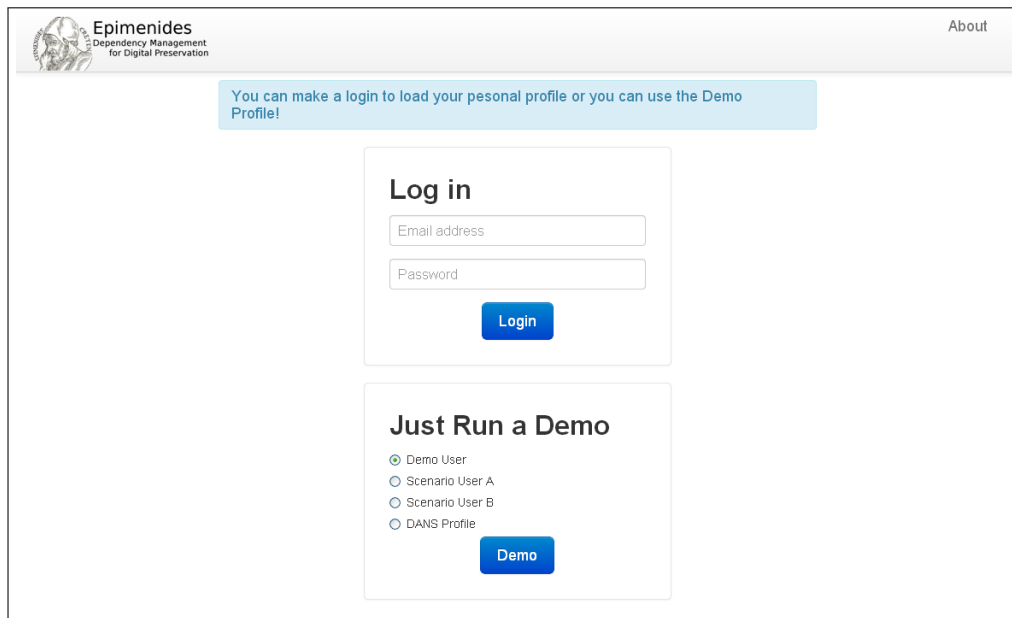
6.7 Screen Dumps of Epimenides

Figure 6.10 shows the first screen of the system, where you can make a login to load your personal profile or you can load some demo profiles.

Figure 6.11 shows the first screen that allows the user to upload a file (atomic or a zipped collection of files). After uploading a file the system analyzes the contents of the zip file and for each of the included files it suggests a task. This is shown in Figure 6.12.

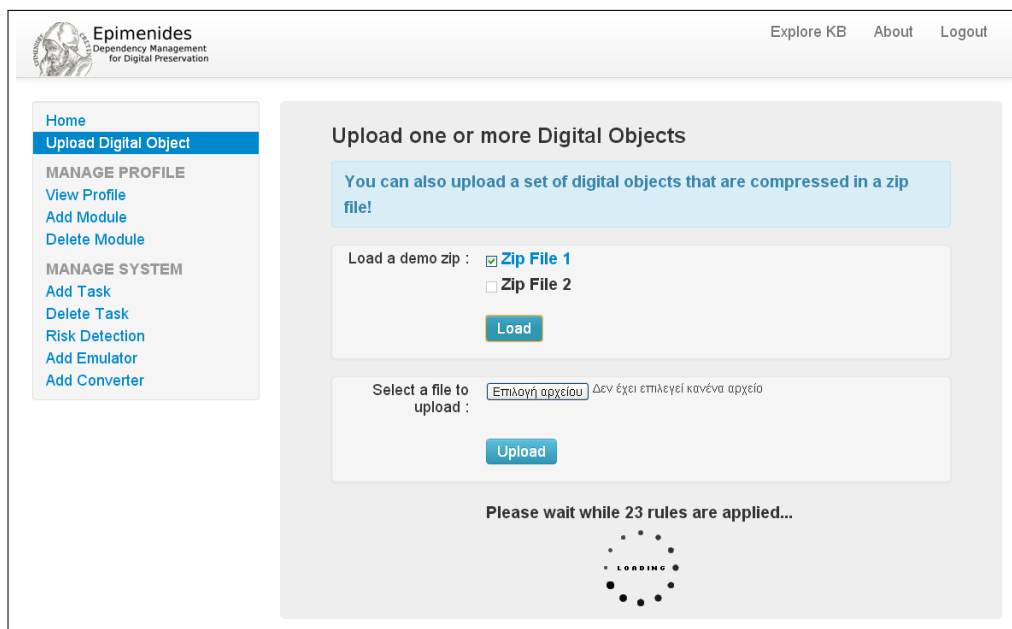
Figure 6.13 shows the results of this analysis. We can see that the first file is in red because the selected task, i.e. Rendenring, cannot be performed over that

file digital object). In contrast, the selected tasks for the other two files can be performed, and for this reason they are marked with green.



The screenshot shows the Epimenides web interface. At the top left is the logo and name 'Epimenides Dependency Management for Digital Preservation'. At the top right is a link 'About'. Below the header is a blue notification box that says 'You can make a login to load your personal profile or you can use the Demo Profile!'. The main content area has two sections. The first section is titled 'Log in' and contains two input fields: 'Email address' and 'Password', followed by a blue 'Login' button. The second section is titled 'Just Run a Demo' and contains four radio buttons: 'Demo User' (selected), 'Scenario User A', 'Scenario User B', and 'DANS Profile', followed by a blue 'Demo' button.

Figure 6.10: Load your personal profile or use a demo profile



The screenshot shows the Epimenides web interface for uploading digital objects. At the top left is the logo and name 'Epimenides Dependency Management for Digital Preservation'. At the top right are links 'Explore KB', 'About', and 'Logout'. On the left side is a sidebar menu with 'Home' and 'Upload Digital Object' (highlighted). Below the menu are two sections: 'MANAGE PROFILE' with links 'View Profile', 'Add Module', and 'Delete Module'; and 'MANAGE SYSTEM' with links 'Add Task', 'Delete Task', 'Risk Detection', 'Add Emulator', and 'Add Converter'. The main content area is titled 'Upload one or more Digital Objects'. It contains a blue notification box that says 'You can also upload a set of digital objects that are compressed in a zip file!'. Below this is a section 'Load a demo zip :' with two checkboxes: 'Zip File 1' (checked) and 'Zip File 2' (unchecked), followed by a blue 'Load' button. Below that is a section 'Select a file to upload :' with a text input field containing 'Επιλογή αρχείου' and a hint 'Δεν έχει επιλεγεί κανένα αρχείο', followed by a blue 'Upload' button. At the bottom, it says 'Please wait while 23 rules are applied...' and shows a loading spinner with the word 'LOADING' in the center.

Figure 6.11: Upload digital objects to check the performability of them

The screenshot shows the Epimenides web application interface. The header includes the logo, the name 'Epimenides', the subtitle 'Dependency Management for Digital Preservation', and links for 'Explore KB', 'About', and 'Logout'. The left sidebar contains a navigation menu with 'Home' and 'Upload Digital Object' (highlighted), followed by 'MANAGE PROFILE' (View Profile, Add Module, Delete Module) and 'MANAGE SYSTEM' (Add Task, Delete Task, Risk Detection, Add Emulator, Add Converter). The main content area is titled 'Select files and Tasks to analyze'. It contains a blue box with instructions: 'The file you have uploaded contains the following Digital Objects. Please check these you want to analyze it and select a task for each one :'. Below this is a list of objects with checkboxes and task dropdowns: 'readme_en.txt' (Readability), 'Rules_Converters2012_JOURNAL.pdf' (Rendering), 'VLC media player.Ink' (There is not a supported task), 'WinEdt.Ink' (There is not a supported task), and 'xampp Start.exe' (Runnability). An 'Uncheck All' button is at the top left of the list, and an 'Analyze' button is at the bottom right.

Figure 6.12: System finds the tasks that usually make sense to apply to the uploaded digital objects

The screenshot shows the 'Results of the Analysis' page in the Epimenides web application. The header and sidebar are identical to the previous screenshot. The main content area is titled 'Results of the Analysis'. It contains a blue box with instructions: 'You can explore the dependencies for each one of the uploaded Digital Objects'. Below this are three colored boxes representing the analysis results: a red box for 'Rules_Converters2012_JOURNAL.pdf' stating 'The task "Rendering" cannot be performed' with an 'Explore Dependencies' button; a green box for 'readme_en.txt' stating 'The task "Readability" can be performed, you already have the dependencies that are needed!' with an 'Explore Dependencies' button; and a green box for 'xampp Start.exe' stating 'The task "Runnability" can be performed, you already have the dependencies that are needed!' with an 'Explore Dependencies' button.

Figure 6.13: Results of analysis

The user can explore the dependencies for each one of the digital objects. For example Figure 6.14 shows what happens if the user clicks to explore the dependencies of the "Rendering" task. We can see all the rules of the selected task that are available in the system. The atoms of each rule are green or red. Green atoms are available in the profile of the user, while the red are not. Moreover the user can click on an atom to explore the dependencies of this atom, so he can see the rules or the facts of this atom.

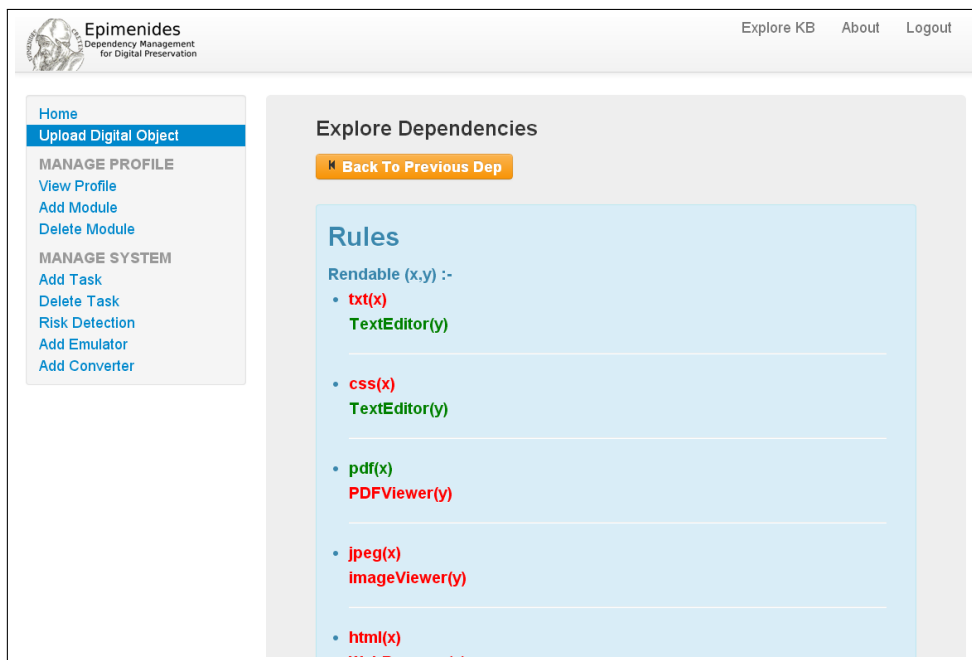


Figure 6.14: Exploring the Dependencies of a Task

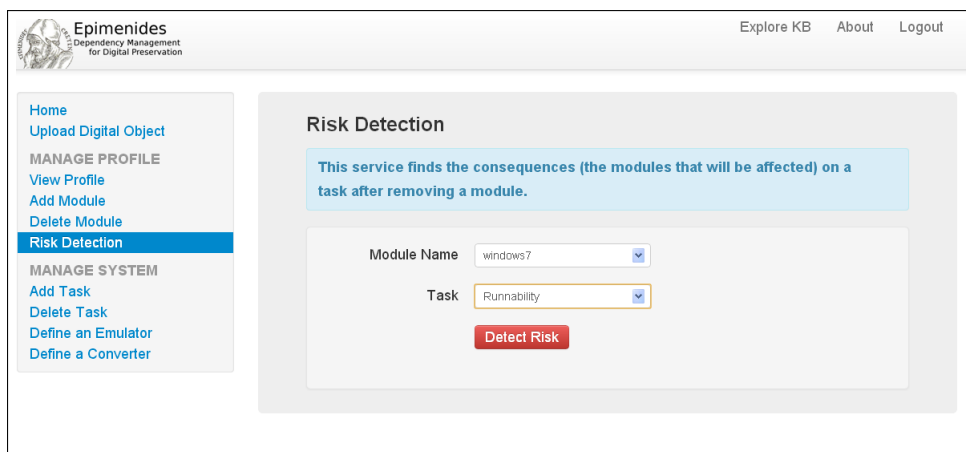
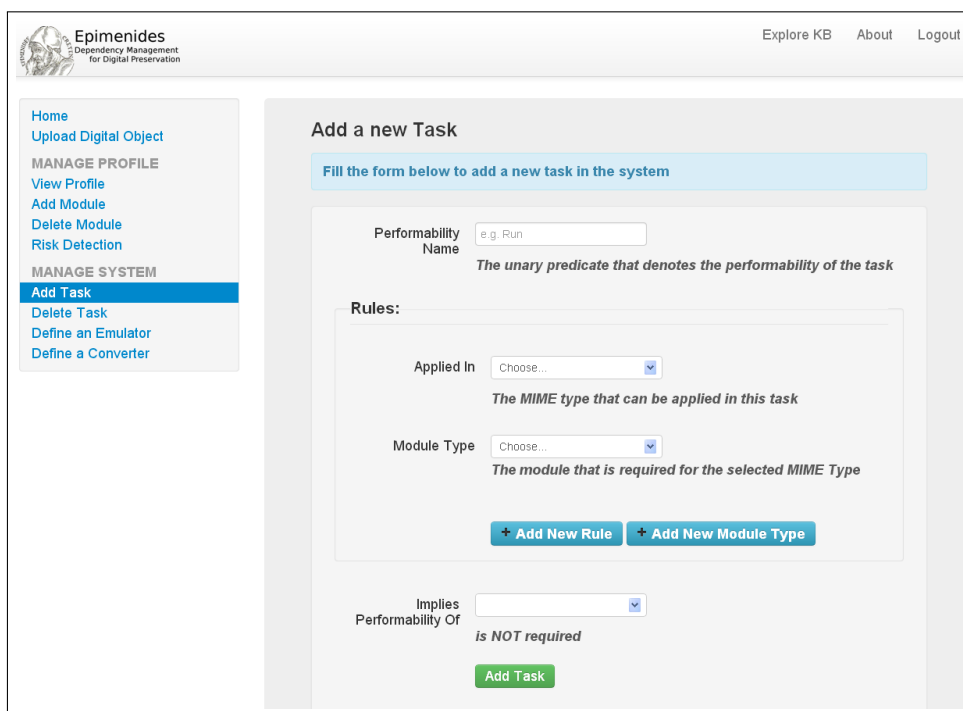


Figure 6.15: Identify the modules that will be affected on a task after removing a module

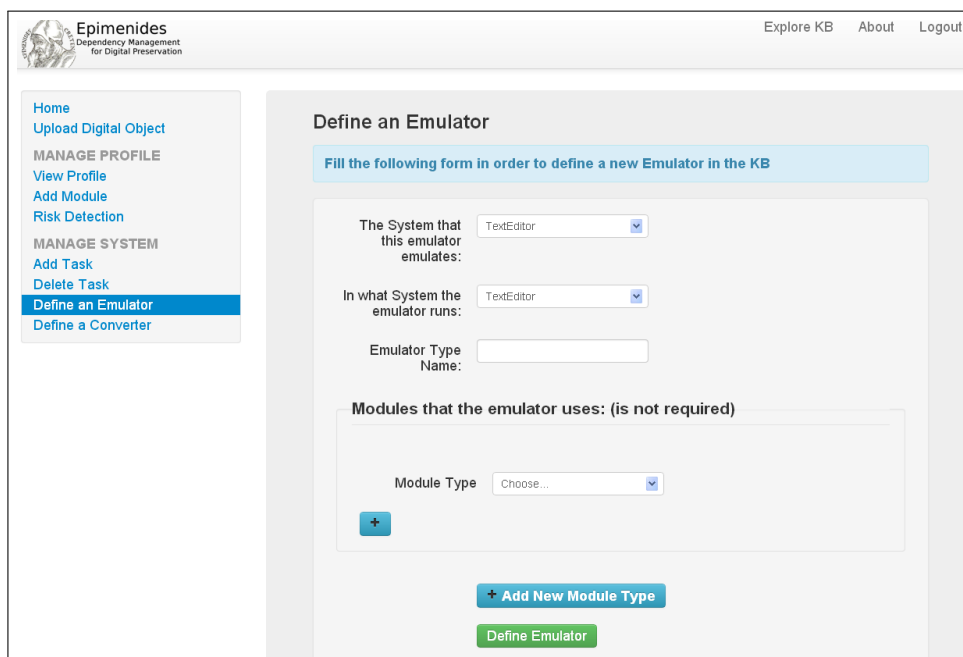
The Consequences of a Hypothetical Loss Service as described in Section 4.2 is supported also by Epimenides. Figure 6.15 shows a screenshot of this service.



The screenshot shows the Epimenides web application interface. The header includes the logo, name, and tagline, along with navigation links. A sidebar on the left contains a menu with categories like 'MANAGE PROFILE' and 'MANAGE SYSTEM'. The main content area is titled 'Add a new Task' and contains a form with the following fields:

- Performability Name:** A text input field with the placeholder 'e.g. Run' and a description: 'The unary predicate that denotes the performability of the task'.
- Rules:** A section containing:
 - Applied In:** A dropdown menu with 'Choose...' and a description: 'The MIME type that can be applied in this task'.
 - Module Type:** A dropdown menu with 'Choose...' and a description: 'The module that is required for the selected MIME Type'.
 - Buttons: '+ Add New Rule' and '+ Add New Module Type'.
- Implies Performability Of:** A dropdown menu with a description: 'is NOT required'.
- Add Task:** A green button at the bottom.

Figure 6.16: Define a new Task



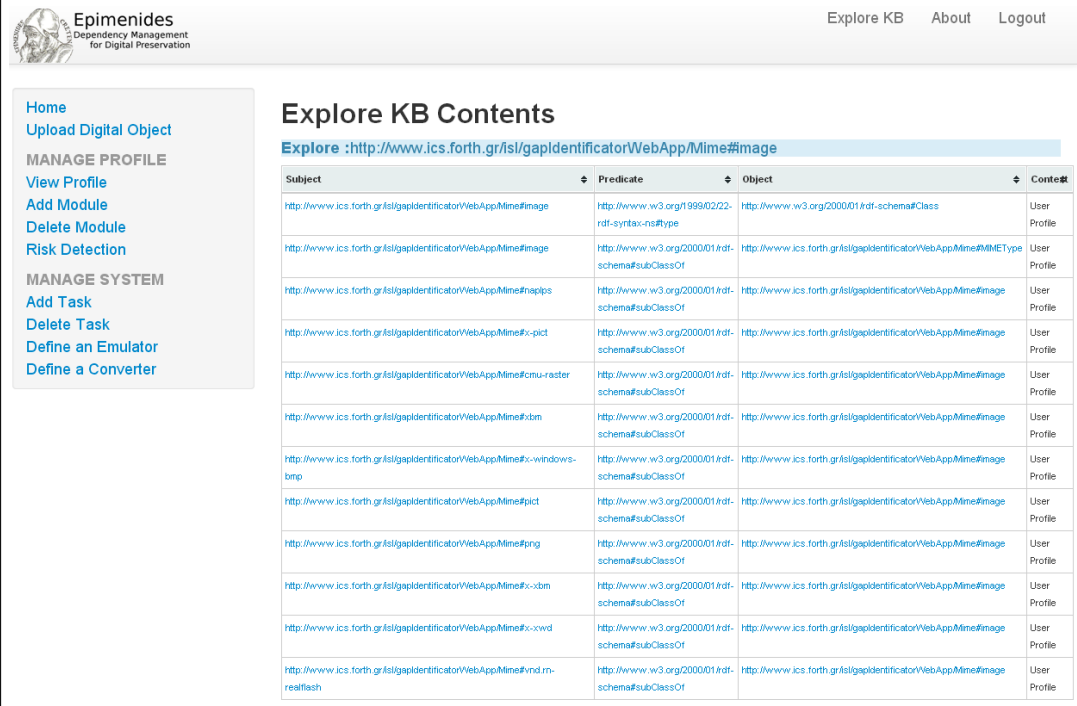
The screenshot shows the Epimenides web application interface. The header and sidebar are identical to the previous figure. The main content area is titled 'Define an Emulator' and contains a form with the following fields:

- The System that this emulator emulates:** A dropdown menu with 'TextEditor' selected.
- In what System the emulator runs:** A dropdown menu with 'TextEditor' selected.
- Emulator Type Name:** A text input field.
- Modules that the emulator uses: (is not required):** A section containing:
 - Module Type:** A dropdown menu with 'Choose...'.
 - Buttons: '+ Add New Module Type' and 'Define Emulator'.

Figure 6.17: Define a new Emulator Type

Figure 6.16 shows the form that a user has to fill in order to define a new task in the system, while Figure 6.17 shows the form for the definition of a new emulator type.

Moreover the user has the ability to explore the contents of its Knowledge Base. This is shown in Figure 6.18.



Epimenides
Dependency Management
for Digital Preservation

Explore KB About Logout

Home
Upload Digital Object
MANAGE PROFILE
View Profile
Add Module
Delete Module
Risk Detection
MANAGE SYSTEM
Add Task
Delete Task
Define an Emulator
Define a Converter

Explore KB Contents

Explore :<http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#image>

Subject	Predicate	Object	Context
http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#image	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class	User Profile
http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#image	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#MIMEType	User Profile
http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#napis	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#image	User Profile
http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#x-pict	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#image	User Profile
http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#cmu-raster	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#image	User Profile
http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#x-bm	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#image	User Profile
http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#x-windows-bmp	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#image	User Profile
http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#pict	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#image	User Profile
http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#png	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#image	User Profile
http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#x-dbm	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#image	User Profile
http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#x-cwd	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#image	User Profile
http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#vnd.m-realflash	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.ics.forth.gr/isl/gapldentifierWebApp/Mime#image	User Profile

Figure 6.18: Explore the contents of the underlying RDF/S triple store

6.8 Evaluating its Usability

We decided to evaluate the usability of the system for investigating if a user can understand the main concepts of the approach by using the system, and how the system per se is usable. For this reason we created a short tutorial for the system⁸, we defined some scenarios⁹ that we asked users to carry out using the system, and we prepared a small questionnaire that the users had to answer after using the system.

⁸<http://users.ics.forth.gr/~kargakis/data/demoUsersGuide.pdf>

⁹<http://users.ics.forth.gr/~kargakis/data/UserExperience.pdf>

Ten users answered this questionnaire with ages ranging from 20 to 30. All of the participants had a computer science background (some of them had a MSc in Computer Science). We can distinguish these users in two groups: the advanced group consisting of 3 users (from APARSEN NoE¹⁰) and the regular ones consisting of 7 users (from Computer Science Department, University of Crete). The advanced users were aware of the dependency management approach, while the regular ones were not. For this reason, and before starting the evaluation, we gave to each regular user a brief tutorial on using the system through examples.

Below we summarize the results the answers of the questionnaire (detailed results are given in Figure 6.19). The results showed that 90% of the participants completed the scenario A, while scenario B was completed from all users (100%). The time to complete both scenarios A and B was less than 6 minutes. From the above we can conclude that **Epimenides** is understandable and easy to use. In questions 5 and 6, all users (100%) answered that the system assisted them in checking the performability of a task and that they better understood why a task can be performed in an existing and unknown profile. This demonstrates the value of the system. Finally, 70% declared that this application is useful for an organization with a big dataset of digital objects. It is also worth noting that no user had ever used any relevant system. At last, a big percentage (90%) of the participants rated with 3 (high) the potential of this approach.

6.9 Query and Reasoning Efficiency

In general performance depends on the capabilities of the adopted triplestore used (for a comparative analysis see [35]). Currently **Epimenides** uses the *Open-Link Virtuoso* RDF triple store. Virtuoso supports backward chaining reasoning, meaning that it does not materialize all inferred facts, but computes them at query level. Its reasoner covers the related entailment rules of `rdfs:subClassOf` and `rdfs:subPropertyOf`. Practically this means that transitive relations (i.e. `subClassOf`, `subPropertyOf`, etc.) are not physically stored in the knowledge base, but they are added to the result set at query answering.

Below we report a few indicative times that concern the current knowledge base

¹⁰<http://www.alliancepermanentaccess.org/>

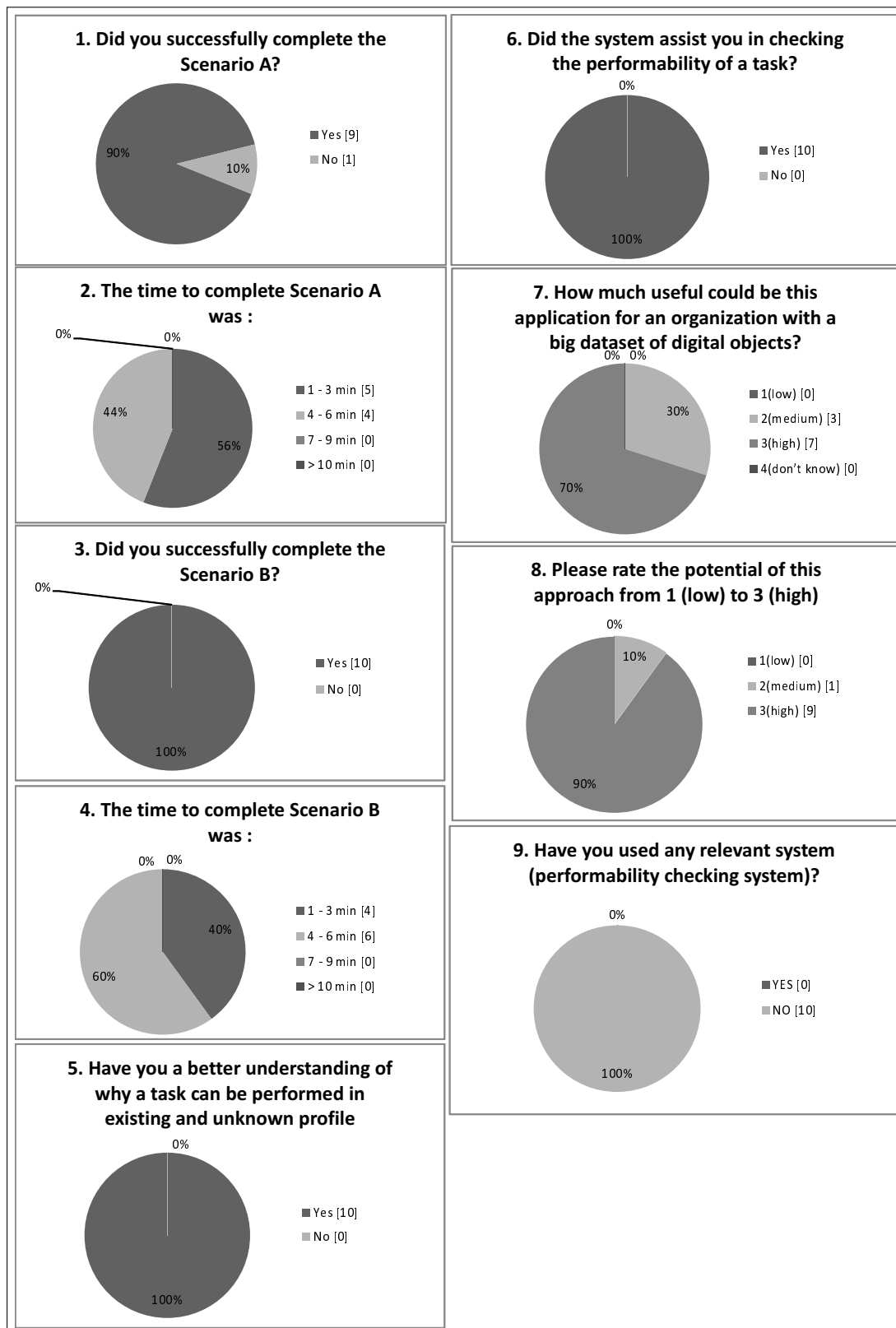


Figure 6.19: Analysis of the responses to the questionnaire

# of Related Modules in KB	# of Fired Rules	Size (in triples) of the Produced <i>OKB</i>	Time for Producing the <i>OKB</i>	Overall Time for the “Consequences of a Hypothetical Loss” Service
3	2	2,227	3.32 sec	6.14 sec
5	6	2,245	7.67 sec	13.21 sec
12	14	2,261	8.90 sec	14.49 sec
18	24	2,299	11.40 sec	19.81 sec

Table 6.1: Some indicative measurements of time

of *Epimenides* which contains 2,304 RDF triples and 53 rules, although efficiency and optimization is not the focus on this thesis¹¹.

Creation Time for the KB. The time to create from scratch the entire KB by loading N-Triple files requires about 4 seconds. The time to add or delete a module is negligible.

Creation Time for the OKB. As we have already mentioned in Section 6.4, the *OKB* is produced by “firing” the rules which in turn produce new triples. The time for the creation of the OKB depends on the number of the fired rules. Table 6.1 shows the time required for creation the OKB from KB, based on the number of the fired rules.

Task Performability Checking. The service for task performability checking relies on the *OKB* and it is reduced to plain query answering. Therefore it is very fast, in average it takes around 37 milliseconds.

Consequences of a Hypothetical Loss. We have also made some experiments regarding the time required for the “Consequences of a Hypothetical Loss” service. This is a composite task that includes all the previous steps, therefore it is appropriate for experimentation or for benchmarking one particular implementation. For the 3 tasks $\{t_1, \dots, t_3\}$ that our KB contains and y (where $y = 3, y = 5$,

¹¹ The experiments were carried out using the Virtuoso 06.01.3127 version, running in a Dual-Core linux machine with 3GB RAM.

$y = 12$ and $y = 18$) in number (related) modules $\{m_1, \dots, m_y\}$, we measured the time required for identifying the consequences on the performability of task t_i after removing the module m_j . Specifically we measured this time for 4 different occasions. Recall that this is achieved by (a) computing the answer of the query $t_i(X)$ (let A be the returned set of elements), (b) deleting m_j from the database, reconstructing the OKB and answering again the query $t_i(X)$ (let B be the returned set of elements), and (c) computing and returning the elements in $A \setminus B$ (they are the ones that will be affected).

The above procedure requires the construction of the OKB twice (one before the computation of A and one before the computation of B). Table 6.1 reports execution times. We can see that the times range from 6 to 20 seconds.

Conclusions. In general we do not expect any difficulty in achieving efficiency (mainly because task performability reduces to plain query answering over the OKB which has already materialized the required information).

Moreover if the adopted triplestore supports custom backwards reasoning, then we will not have to produce OKB, which is the most expensive task.

Chapter 7

Applicability

7.1 On Applicability

We have already seen (in Chapter 1) an example of how our approach can be applied in software. In brief a user that holds the modules that are shown in Figure 7.1a, could run `game.pas` on his mobile phone by first converting the Pascal code to C++, then compiling the C++ code, and finally by running over the emulator the executable yielded by the compilation (these series of transformation/emulations are shown in Figure 7.1b). We should however clarify that the proposed approach is not confined to software. Various services that concern documents and datasets can also be captured. Below we describe some examples in more detail.

Consider that the same user (of the previous example) received the document `secret.doc`. Also consider that recently has bought the `MicrosoftOfficeWord.exe` for Windows OS and a converter from doc to pdf (`doc2pdf`) and he is wondering if he can render this file. Defining the rendering task with a proper way, the reasoning approach can infer that this is possible through various ways :

- by running the `MicrosoftOfficeWord.exe` on his laptop,
- by running the `SuiteOffice` of his smart phone,
- by running over the W4A the `MicrosoftOfficeWord.exe` on his mobile phone,
- converting the `secret.doc` to a pdf file, and then run the pdf Viewer (`SuiteOffice`) in his smart phone.

Note that we suppose that Android OS has as preinstalled : a Word Office software (`SuiteOffice`) and a pdf viewer software (`SuiteOffice`).

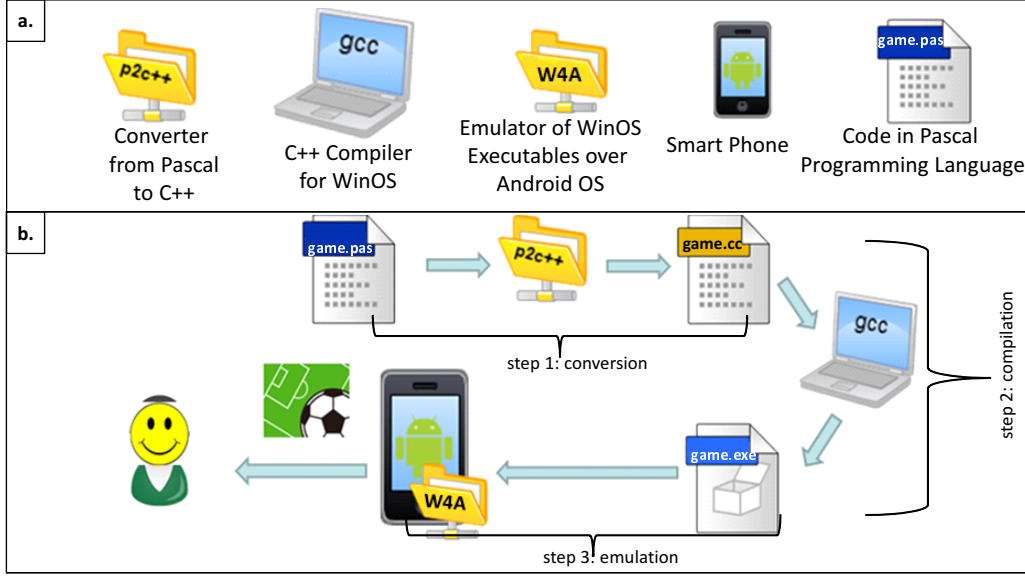


Figure 7.1: Our approach in a software example

For the case of datasets, consider that we want to preserve datasets containing experimental results and would like to preserve their provenance. Suppose that for us provenance means ability to answer questions of the form: who derived the dataset, when this dataset was derived, how it was derived? We can model provenance as a task (that has dependencies) and we can use the dependency reasoning approach for in a way that enables checking for which datasets we have provenance and for which we have not. We could also exploit the reasoning services in order to discover provenance information that was not evident (e.g. result of tools that extract embedded metadata).

7.2 Ways to Offer the Dependency Management Approach

There are more than one ways to apply the approach presented in this thesis. Below we discuss two main approaches.

- **As a System.** Here the idea is to have one dedicated system, adapted to the needs, practices and other operational system of the organization/archive. Systems like *Epimenides* fall into this case. Alternatively, one could “inject”

the dependency management approach to existing repository management systems. For instance, the dependency management approach could be implemented by extending the Fedora repository. This could be quite straightforward since the Fedora stores metadata using RDF/S and the set of relations that can be used for connecting objects is not limited (more about Fedora in the next Section)

- **As a Service.** The proposed approaches could be offered as a service by third party providers, e.g. by a provider of cloud services who apart from offering storage services, it offers various virtualization services and uses the methodology and techniques described in this document for realizing them.

Clearly other tools and datasets can also contribute to an operational application of the approach. Section 7.4 describes such tools and datasets.

7.3 Application by Extending an Existing Repository (Fedora)

Fedora¹ is a widely used repository management system for digital objects that provides tools and interfaces for the creation, ingest, management, and dissemination of content. The key abstraction, is the Fedora Digital Object (FDO). A FDO has an identifier (PID), Dublin Core metadata, and Datastreams (the actual content). A Datastream can be of any MIME-type and it can be managed locally (in the Fedora repository), or by external data sources (in that case it referenced by its URL). FDOs can be connected through relationships forming a network of digital objects, and these relationships are stored as metadata in digital objects within special Datastreams. The Fedora repository service automatically indexes all the relationships creating a graph of all the objects in the repository and their relationships to each other. The user can then make queries (e.g. SPARQL queries) to this graph and take results of the repository content. Fedora has also the ability to associate the data in a FDO with Web services to produce dynamic disseminations where a dissemination is a view of an object produced by a service operation (i.e. a method invocation) that takes as input one or more datastreams of the object.

¹<http://fedora-commons.org/>

Our approach could be implemented by extending the Fedora repository. This could be quite straightforward since the Fedora stores metadata using RDF/S and the set of relations that can be used for connecting objects is not limited. One way could be to extend the Fedora with a service that takes as input the MIME-type of the contents (datastreams) from the FDOs. This service using those MIME-type and having a basic mapping between the MIME-types and the tasks (e.g the MIME-type `application/msword` must checked for the task `render`) can automatically define the required dependencies. The KB that stores these could be the same with that of Fedora, or an external one. Of course the administrator of the repository could define various other tasks and dependencies using the approach that we have described.

7.4 Related Datasets and Tools

7.4.1 PreScan

PreScan [9] is a tool developed in the context of the EU project CASPAR². It can aid the ingestion of metadata. Figure 7.2 sketched the process that it carries out.

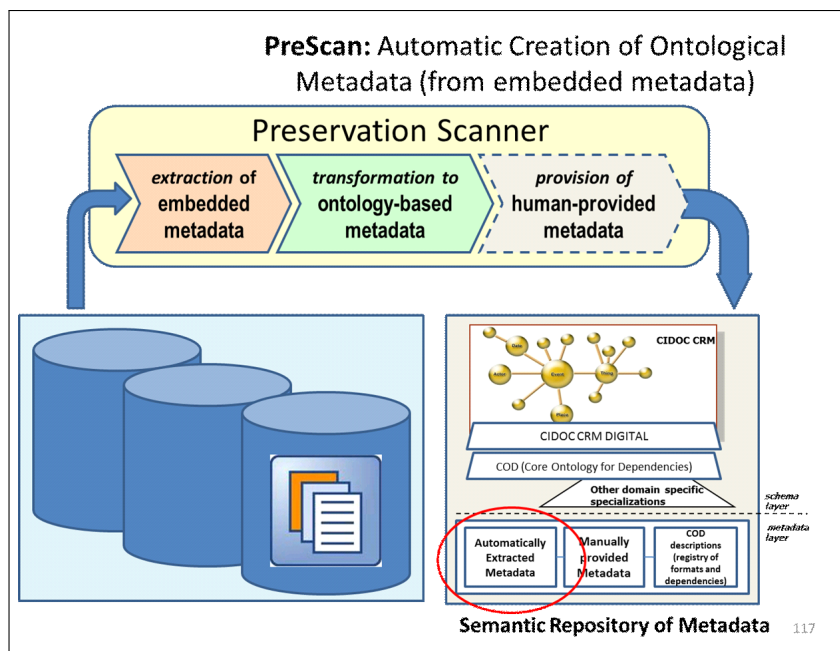


Figure 7.2: The system PreScan

²<http://www.casparpreserves.eu/>

In brief, this tool can scan the file system, extract the embedded metadata from the files, and transform them to RDF using the desired RDF schema. In the sequel, the resulting metadata could feed the Knowledge Base of *Epimenides*.

7.4.2 The PRONOM Registry and its Contents

PRONOM³ is an on-line information system about data file formats and their supporting software products. Originally developed to support the accession and long-term preservation of electronic records held by the National Archives. PRONOM holds information about software products, and the file formats which each product can read and write.

Linked Data PRONOM Lab⁴ plans to make the registry data available in a Linked Open Data format. They created an RDF triplestore and a SPARQL Endpoint⁵ is available. Also a draft vocabulary specification and accompanying documentation in RDF are available⁶. Table 7.1 shows the used RDF properties.

In comparison to our approach PRONOM is less powerful. PRONOM does not model the notion of task. Moreover, the notion of converter and emulator is not covered.

However, we could exploit some information from the PRONOM registry in order to enrich the Knowledge Base of our prototype. Specifically, as we have seen in table 7.1, there are some common properties (i.e. extension and mime Type). For example from the extension of a file we can retrieve extra information from PRONOM registry by running the following SPARQL query:

```
select ?puid ?mime ?description ?developer where {
  ?s <http://reference.data.gov.uk/technical-registry/extension> "doc".
  ?s <http://reference.data.gov.uk/technical-registry/PUID> ?puid .
  ?s <http://reference.data.gov.uk/technical-registry/MIMETYPE> ?mime.
  ?s <http://purl.org/dc/elements/1.1/description> ?description.
  ?s <http://reference.data.gov.uk/technical-registry/developedBy> ?developer
}
```

The Persistent Unique Identifier (PUID) shown in the previous query is an extensible scheme for providing persistent, unique and unambiguous identifiers for records

³<http://www.nationalarchives.gov.uk/PRONOM/Default.aspx>

⁴<http://labs.nationalarchives.gov.uk/wordpress/index.php/2011/01/linked-data-and-pronom>

⁵<http://test.linkeddatapronom.nationalarchives.gov.uk/sparql/endpoint.php>

⁶<http://test.linkeddatapronom.nationalarchives.gov.uk/vocabulary/pronom-vocabulary.htm>

Properties
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://www.w3.org/2000/01/rdf-schema#label
http://reference.data.gov.uk/technical-registry/version
http://www.w3.org/2004/02/skos/core#altLabel
http://reference.data.gov.uk/technical-registry/formatType
http://purl.org/dc/elements/1.1/description
http://reference.data.gov.uk/technical-registry/byteOrder
http://reference.data.gov.uk/technical-registry/releaseDate
http://reference.data.gov.uk/technical-registry/withdrawnDate
http://reference.data.gov.uk/technical-registry/MIMETYPE
http://reference.data.gov.uk/technical-registry/PUID
http://reference.data.gov.uk/technical-registry/extension
http://reference.data.gov.uk/technical-registry/internalSignature
http://reference.data.gov.uk/technical-registry/byteSequence
http://reference.data.gov.uk/technical-registry/byteSequencePosition
http://reference.data.gov.uk/technical-registry/byteSequenceOffset
http://reference.data.gov.uk/technical-registry/byteString
http://reference.data.gov.uk/technical-registry/UTI
http://reference.data.gov.uk/technical-registry/developedBy
http://reference.data.gov.uk/technical-registry/maxByteSequenceOffset
http://reference.data.gov.uk/technical-registry/supportedBy
http://reference.data.gov.uk/technical-registry/XPUID
http://www.w3.org/2004/02/skos/core#note
http://reference.data.gov.uk/technical-registry/lossiness
http://reference.data.gov.uk/technical-registry/WAVE_Format_GUID
http://reference.data.gov.uk/technical-registry/compressionDocumentation
http://reference.data.gov.uk/technical-registry/mediaFormat

Table 7.1: The RDF properties of PRONOM

in the PRONOM registry. A unique PUID is assigned to each registry entry of the PRONOM.

However, the PRONOM registry contains only 101 mime types, while the Knowledge Base of **Epimenides** already contains 647 different mime types, therefore the value of PRONOM is limited.

Moreover, there is a RDF repository (P2-Registry [36]) that links the PRONOM registry data with the DBpedia data. The P2-Registry is available at: <http://p2-registry.ecs.soton.ac.uk/> . This registry provides open access to all the data contained within, as well as services including a SPARQL endpoint⁷ and RESTful HTTP services. Data is currently available in XML and RDF formats, an HTML interface is not currently proposed other than to offer an explanation of the services available.

7.4.3 Catalogues of Existing Converters and Emulators

We tried to find if there is a place where the capabilities of existing transformers and emulators are described. Ideally we would like to find a format that allows representing and exchanging such information. If such format existed, that would allow us design a method that takes as input such a description and produces the required facts and rules of our approach.

Unfortunately, we have not managed to find such a registry. There is one page at wikipedia⁸ that lists software and hardware that emulate computing platforms. Their description is textual (in HTML tables). It contains 339 emulators, where for each one the page provides its: *Name*, *Actual Version*, *System* (e.g. x86 PC, x86-64 PC), *Platform* (e.g. Windows), *Licence* and a link to the *official site* of the emulator.

7.5 Case Study: DANS

In the context of APARSEN NoE we have also conducted a case study for the case of DANS (Data Archiving and Networked Services, NL)⁹. DANS aims at promoting sustained access to digital research data. For this purpose, it encourages researchers

⁷<http://p2-registry.ecs.soton.ac.uk/SPARQL/>

⁸http://en.wikipedia.org/wiki/List_of_computer_system_emulators

⁹<http://www.dans.knaw.nl/en>

to archive and reuse data in a sustained manner, e.g. through the online archiving system EASY (<http://easy.dans.knaw.nl>). DANS also provides access, via NARCIS (<http://www.narcis.nl>), to scientific datasets, e-publications and other research information in the Netherlands. Apart from these, the institute provides training and advice, and performs research into sustained access to digital information.

In collaboration with DANS, we have defined a number of scenarios that indicate where and how the dependency management approach could be used. The analysis yielded five main scenarios.

In brief the desired (for DANS) tasks are related with the acceptable/preferred formats, and with the runability of DANS software (including computability of checksums).

7.5.1 Scenario 1: Checking File Format Compatibility (compliance or migratability) with Acceptable/Preferred File Formats during Ingestion

Description: For a number of data types (tables, text, images, etc.), specific file formats are considered as durable at least into the near future. DANS maintains a list of acceptable and preferred formats. These lists are the basis for file format migration activities. The list that DANS currently uses is available¹⁰.

Applicability: If the converters (or emulators) that are in use by DANS for carrying out the migration activities, are registered in a system like Epimenides, then the system can be exploited not only for checking whether a newly ingested file is in an acceptable/preferred format, but also for checking whether it is migratable to one preferred or acceptable format using the migration/emulation software that DANS uses and has registered.

Implementation: To realize this scenario, one has to define a profile (say `profile_DANS`) that consists of:

- a. The list containing the software that DANS uses for managing a file having an acceptable/preferred file format (e.g. AcrobatReader for rendering PDF

¹⁰ Taken from <http://www.dans.knaw.nl/sites/default/files/file/EASY/DANS%20preferred%20formats%20UK%20DEF.pdf>

files, VLC for playing mpg/mpeg/mp4/avi/mov files). At least one software per format is required.

- b. For each file type in the list of acceptable/preferred list, a task has to be associated (the one usually applicable on such file types) and the dependencies for that task have to be delivered in a way so that they are satisfied by the list of software described in [a] (e.g. `Render(X) :- pdfFile(X), pdfViewer(Y)`).
- c. The list of tools that DANS uses for migration/conversion purposes (e.g. `docxToPdfConverter(doc2pdf)`).

Use Case: After having done the steps that were just described, the end user (or archivist) could just use the system. Whenever he uploads a file, the system prompts the applicable task and directly informs the user if it is in an acceptable format or migratable to an acceptable format using the software that DANS has.

Without such facility it is difficult for a curator to determine that (a) an archived dataset is formatted in a durable format and (b) to have an overview of the applicable file format migration procedures that can be carried out to convert a file into a preferred file format (given the fact that the list of preferred file formats will change over time as file formats might become obsolete).

7.5.2 Scenario 2: Updating the List of Preferred/Acceptable Formats and Detecting the Consequences of Obsolete Formats

Description: As the usability and durability of file formats tend to change over time, for DANS it is important to periodically monitor and assess the applicability of the list of preferred formats and if it is necessary to replace a file format that became obsolete with a new one. Also new preferred formats can be introduced in the list. Specifically, say every year, the specifications on the list of preferred file formats have to be assessed based on a number of criteria (e.g. discussions in literature, consensus of organizations that provide guidelines in this field, etc.)

Applicability:

- a. To add a new format in the list of acceptable/preferred file formats, the archivist can just register it to the knowledge base of Epimenides. The check

performed at ingestion time will then function as expected (i.e. in accordance with the revised list of acceptable formats).

- b. Before deleting a file format (or managing software) from the list of acceptable/preferred file formats (or available software respectively), the archivist can check the impact of that deletion, i.e. the impact that this deletion will have on the performability of tasks over the archived files. Recall the discussion on Section 5.2 about the “Consequences of a Hypothetical Loss”.
- c. To delete a file format (or managing software) from the list of acceptable/preferred file formats (or available software respectively), the archivist can just delete the corresponding entries from the system. After doing so, the checking at ingestion time will function as expected, i.e. in accordance with the revised list of acceptable formats.

Without such services it is difficult to identify all the consequences of file format’s obsolescence. It is also difficult to identify what will happen if managing software¹¹ is lost or will become obsolete.

7.5.3 Scenario 3: Assistance in Planning and Performing Migration to Acceptable/Preferred File Formats

Description: Research datasets are formatted in a number of formats as submitted to the data archive by the depositors. The data archive stores and manages these datasets in the format as submitted by executing so-called “bit-preservation” (more about bit preservation in a next scenario). The data archive archives all formats but only commits itself to the long-term usability of file format that are formatted according to so-called preferred formats, described in the previous scenarios.

In two situations a file format migration is required: (1) as part of the *ingest procedure* files not formatted according to the preferred file format are migrated to a suitable preferred file format. (2) in case in the future a *preferred file format becomes obsolete* the files have to be migrated to this new format.

The migration process requires tools. Quality features of these tools are: speed, accuracy, level of completeness, and usability of the tool.

¹¹Software that is able to convert to/from a preferred file format

Applicability & Implementation: The dependency management approach can show to the archivist whether a file format migration is possible using the software that DANS has (recall Scenario 1). Also since a migration can be performed with different tools (or execution plans in general), the proposed system can assist the archivist by showing to him/her, the possible actions/tools and this can be achieved by exploring the dependencies that the system offers (recall the screens of the system that offer exploration services).

Without such services it is difficult for a human to identify all possible migration plans.

7.5.4 Scenario 4: Checking the Preservation of the Software

Description: Despite the fact that research data archives are aimed at the durable access of datasets, there are cases where specific software is required to be able to use the datasets. For such cases, activities have to be undertaken to guarantee that this software is usable over time. Software preservation involves much more dependencies, than research data preservation (e.g. changing operating systems, proprietary source code, etc.). Research data archives currently have no general accepted software preservation strategy.

Applicability & Implementation: The examples of the current thesis have demonstrated this with various examples (recall the task of runability and compilability).

7.5.5 Scenario 5: Bit Preservation (ability to test corruption)

Description: The bit preservation scenario involves activities to guarantee that digital objects do not become corrupted. This means not one bit is changed over time. Thus the integrity of the data objects is guaranteed. This can be achieved by creating checksums on the occasion where the digital objects are ingested in the data archive and periodically check whether the checksum is still valid. Dependencies in the scenario are the strength of the checksum procedures and the time interval the checksum is checked as part of the bit preservation activities.

Applicability & Implementation: If checksums are supposed to be used for

ensuring that the data have not been corrupted, then an archive can model as task the computation of checksums for being sure that in the future the archiving organization will be able to recompute them and compare them with the stored ones. Note that there are several tools for computing checksums¹². We can say that this is a special case of scenario 4.

Without our approach it is difficult and time consuming to plan software migration.

7.5.6 Consolidation of the Scenarios

Here we consolidate the key points of the above scenarios and Table 7.5 describes them using as gnomon the steps of the methodology introduced in Section 3.2

7.6 Layering Tasks

We should stress that the modeling approach presented in this thesis allows modeling and organizing tasks *hierarchically*. This is quite natural, and we have seen that the community and the literature many times attempts to provide a kind of layering. Below we describe, quite generally, some tasks. In some cases, the more we go down to the list, the more complex the tasks become, i.e. some of these tasks rely on the ability of performing other tasks.

Ability to:

- Retrieve the bits: Ability to get a particular set of stored bits.
- Access: Ability to retrieve the bits starting from an identifier (e.g. a persistent identifier)
- Render: Given a set of bits, ability to render them using the right symbol set (e.g. as defined in [24]) for creating the intended sensory impression.
- Run: Ability to run a program in a particular computer platform.
- Search: Ability to find a digital object. Search ability can be refined based on the type of the object (doc, structured, composite) and its searchable part (contents, structure, metadata).
- Link: Ability to place a digital object in context and exploit it. This may require combining data across different sources.

¹²http://en.wikipedia.org/wiki/Checksum#Checksum_tools.

General Step	Specialization for the case of DANS
1. Identify the desired tasks and objectives	<p>The desired tasks are:</p> <ul style="list-style-type: none"> a. those related to the list of the acceptable/preferred formats, e.g. <code>render</code> (for pdf, txt, pictures), <code>play</code> (for video, audio), <code>getTheRelationalModel</code> (for spreadsheets, databases), etc. b. those related to the runability of DANS software (including computability of checksums).
2. Model them and their dependencies (check hierarchy)	<ul style="list-style-type: none"> a. Using the list of software described in Scenario 1[a] (Section 7.5.1). Moreover the dependencies of the runability of the tools that DANS uses for migration have to be modeled. b. Model the software dependencies that are required for running the software that DANS uses. <p>In general the modeling required is quite simple, analogous to the examples given in the thesis.</p>
4. Identify Ways to capture dependencies (manual, auto, ...)	The file types are detected automatically (when one uses the upload feature of Epimenides). For applying this approach in a big collections of files, various tools could be used for automating this process (more in Section 7.2). Surely, in an operational setting the proposed functionality could extend or complement the functionality of the ingestion procedures of the systems that DANS currently uses.
5. Customize use and exploit the dependency services	For demonstration purposes this can be done using the Epimenides , i.e. no need for customization or integration with the other systems of DANS. However, in an operational setting the processes and systems of DANS should be considered. Applicability is discussed in more detail in 7.2.
6. Evaluate	This can be done using the Epimenides .

Table 7.2: Application of the Methodology for the case of DANS

- Assert Quality: Ability to answer questions of the form: What is its value of this digital object, is it authentic?
- Get Provenance: Ability to answer the corresponding questions (who, when, how).
- Assert Authenticity: (based on provenance, etc)
- Reproduce: Ability to reproduce a scientific result. This is crucial for e-Science.
- Update: Ability to update and evolve a digital object.
- Upgrade/Convert/Transform: Ability to upgrade a digital object (e.g. to a new format), or convert its form.

Chapter 8

Concluding Remarks

As the scale and complexity of information assets and systems evolves towards overwhelming the capability of human archivists and curators (either system administrators, programmers and designers), it is important to offer services that can check whether it is feasible to perform a task over a digital object. For example, a series of conversions and emulations could make feasible the execution of software written in 1986 software on a 2013 platform. The process of checking whether this is feasible or not could be too complex for a human and this is where advanced reasoning services could contribute, because such services could greatly reduce the human effort required for periodically checking (monitoring) whether a task on a digital object is performable.

Towards this vision, in this thesis we have advanced past rule-based approaches for dependency management for capturing *converters* and *emulators*, and we have demonstrated that the proposed modeling enables the desired automatic reasoning regarding task performability.

We proposed a methodology, we described the services which are more useful for the needs of digital preservation, and we showed how we can offer such services in an RDF/S implementation. For the latter, we distinguished various policies about how to tackle evolution and inference, and some limitations of RDF/S. Also we showed that our modeling approach can model real converters and emulators. In the sequel, we described a dataset and a prototype system, called **Epimenides**, that we have build based on the proposed approach which proves the technical feasibility, while the evaluation of its usability showed the underlying concepts can

be easily understood by users. Although the knowledge base of the prototype system currently represents only some indicative tasks, it can demonstrate the benefits of the proposed approach. Finally we described how the proposed approach can be applied in other existing systems and tools.

We could say that the long term vision in the digital preservation is the *virtualization of the basic preservation tasks*. Just like the virtualization of *storage* that is currently offered by the cloud have made the life easier for the organizations that have to keep stored content, the virtualization of *rendering* and *software execution* would be an important contribution to digital preservation, and significant relief for the responsible organizations. To realize this virtualization, and preserve the performability of these tasks as operating systems, protocols, format change, the provider of such services needs a repository and services like those that we have described in this thesis.

From the technical side, an objective for future research is to develop quality-aware reasoning for enabling quality-aware preservation planning. For example consider that we can render an image through a conversion, but the result is in a lower resolution than native. Other issues for future research include gap visualization methods and update requirements.

Finally it will be worth to investigate the usage of other logical languages. For example the event calculus language [52] can be exploited in order to model and extend our approach.

Bibliography

- [1] Y. Tzitzikas, “Dependency management for the preservation of digital information,” in *Procs of the 18th Intern. Conf. on Database and Expert Systems Applications, DEXA’2007*, Regensburg, Germany, September 2007.
- [2] Y. Tzitzikas and G. Flouris, “Mind the (intelligibly) gap,” in *Procs of the 11th European Conference on Research and Advanced Technology for Digital Libraries, ECDL’07*. Budapest, Hungary: Springer-Verlag, September 2007.
- [3] Y. Tzitzikas, Y. Marketakis, and G. Antoniou, “Task-based Dependency Management for the Preservation of Digital Objects using Rules,” in *Procs of 6th Hellenic Conf. on Artificial Intelligence, SETN-2010*, Athens, Greece, 2010.
- [4] G. E. David, *Advanced Digital Preservation*. Springer, 2010.
- [5] C. Becker and A. Rauber, “Decision criteria in digital preservation: What to measure and how,” *JASIST*, vol. 62, no. 6, pp. 1009–1028, 2011.
- [6] M. Waller and R. Sharpe, *Mind the Gap: Assessing Digital Preservation Needs in the UK*. The Digital Preservation Coalition, 2006.
- [7] S. Strodl, C. Becker, R. Neumayer, and A. Rauber, “How to choose a digital preservation strategy: Evaluating a preservation planning procedure,” in *Proceedings of the 7th ACM IEEE Joint Conference on Digital Libraries (JCDL 2007)*, 2007, pp. 29–38.
- [8] Y. Marketakis and Y. Tzitzikas, “Dependency Management for Digital Preservation using Semantic Web technologies,” *International Journal on Digital Libraries*, vol. 10, no. 4, 2009.
- [9] Y. Marketakis, M. Tzanakis, and Y. Tzitzikas, “PreScan: Towards Automating the Preservation of Digital Objects,” in *Procs of the International Conference on Management of Emergent Digital Ecosystems MEDES’2009*, Lyon, France, October, 2009.
- [10] S. Granger, “Digital preservation & emulation: from theory to practice,” *ICHIM (2)*, pp. 289–296, 2001.

- [11] D. Waters and J. Garrett, "Preserving Digital Information Report of the Task Force on Archiving of Digital Information," in *Commissioned by the Commission on Preservation and Access and the Research Libraries Group, Inc., Washington DC: Commission on Preservation and Access.*, 1996.
- [12] S. Granger, "Emulation as a digital preservation strategy," 2000, corporation for National Research Initiatives.
- [13] F. Bellard, "QEMU, a fast and portable dynamic translator," in *Procs of the USENIX Annual Technical Conference, FREENIX Track*, 2005, pp. 41–46.
- [14] J. Van der Hoeven, B. Lohman, and R. Verdegem, "Emulation for digital preservation in practice: The results," *International Journal of Digital Curation*, vol. 2, no. 2, 2008.
- [15] B. Lohman, B. Kiers, D. Michel, and v. d. J. Hoeven, "Emulation as a Business Solution: the Emulation Framework," in *Procs of the 8th International Conference on Preservation of Digital Objects (iPres'2011)*, 2011.
- [16] D. von Suchodoletz, K. Rechert, J. van der Hoeven, and J. Schroder, "Seven steps for reliable emulation strategies—solved problems and open issues," in *7th Intern. Conf. on Preservation of Digital Objects (iPRES2010)*, 2010, pp. 19–24.
- [17] K. Rechert, D. von Suchodoletz, and R. Welte, "Emulation based services in digital preservation," in *Procs of the 10th annual joint conference on Digital libraries.* ACM, 2010, pp. 365–368.
- [18] R. A. Lorie, "Long term preservation of digital information," in *Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries*, ser. JCDL '01. New York, NY, USA: ACM, 2001, pp. 346–352. [Online]. Available: <http://doi.acm.org/10.1145/379437.379726>
- [19] J. R. Van Der Hoeven, R. J. Van Diessen, and K. Van Der Meer, "Development of a universal virtual computer (uvc) for long-term preservation of digital objects," *J. Inf. Sci.*, vol. 31, no. 3, pp. 196–208, Jun. 2005. [Online]. Available: <http://dx.doi.org/10.1177/0165551505052347>
- [20] W. Bergmeyer, "The KEEP Emulation Framework," in *Proceedings of the 1st International Workshop on Semantic Digital Archives (SDA 2011)*, 2011.
- [21] S. Ceri, G. Gottlob, and L. Tanca, "What You Always Wanted to Know About Datalog (And Never Dared to Ask)," *IEEE Transactions on Knowledge and Data Engineering*, vol. I, no. 1, 1989.
- [22] D. Elenius, D. Martin, R. Ford, and G. Denker, "Reasoning about Resources and Hierarchical Tasks Using OWL and SWRL," in *Procs of the 8th International Semantic Web Conference (ISWC'2009)*, 2009.

- [23] M. Theodoridou, Y. Tzitzikas, M. Doerr, Y. Marketakis, and V. Melessanakis, "Modeling and Querying Provenance by Extending CIDOC CRM," *J. Distributed and Parallel Databases (Special Issue: Provenance in Scientific Databases)*, 2010.
- [24] M. Doerr and Y. Tzitzikas, "Information Carriers and Identification of Information Objects: An Ontological Approach)," 2012, coRR, Digital Libraries, arXiv: 1201.0385v1 [cs.DL].
- [25] R. Fikes and N. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, vol. 2, no. 3-4, pp. 189–208, 1972.
- [26] Y. Tzitzikas, Y. Marketakis, and Y. Kargakis, "Conversion and Emulation-aware Dependency Reasoning for Curation Services ," in *Proceedings of the 9th Annual International Conference on Digital Preservation (iPres2012)*, 2012.
- [27] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosz, and M. Dean, "Swrl: A semantic web rule language combining owl and ruleml," May 2004, (<http://www.w3.org/Submission/SWRL/>).
- [28] D. L. McGuinness and F. van Harmelen, "Owl web ontology language overview," 2004, (<http://www.w3.org/TR/owl-features/>).
- [29] O. Erling and I. Mikhailov, "RDF Support in the Virtuoso DBMS," in *Procs of 1st Conference on Social Semantic Web*, 2007.
- [30] G. Manjunath, C. Sayers, D. Reynolds, V. KS, S. K. Mohalik, B. R, J. L. Recker, and M. Mesarina, "Semantic Views for Controlled Access to the Semantic Web," in *Workshop on Semantic Web for Collaborative Knowledge Acquisition (SWeCKA07)*, 2008.
- [31] R. Volz, D. Oberle, and R. Studer, "Implementing views for light-weight web ontologies," in *SAC2003*, 2003.
- [32] A. Magkanaraki, V. Tannen, V. Christophides, and D. Plexousakis, "Viewing the semantic web through rvl lenses," in *ISWC03*, 2003.
- [33] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl, "Rql: A declarative query language for rdf." ACM Press, pp. 592–603.
- [34] W. Le, S. Duan, A. Kementsietsidis, F. Li, and M. Wang, "Rewriting queries on sparql views," in *Proceedings of the 20th international conference on World wide web*, ser. WWW '11. New York, NY, USA: ACM, 2011, pp. 655–664. [Online]. Available: <http://doi.acm.org/10.1145/1963405.1963497>
- [35] B. Haslhofer, E. Momeni Roochi, B. Schandl, and S. Zander, "Europeana RDF store report," 2011.

- [36] D. Tarrant, S. Hitchcock, and L. Carr, “Where the Semantic Web and Web 2.0 meet format risk management: P2 registry,” in *In Procs of the 6th Intern. Conf. on Preservation of Digital Objects (iPres 2009)*, 2009.
- [37] A. Shaon, D. Giaretta, S. Crompton, E. Conway, B. Matthews, F. Marelli, U. D. Giammatteo, Y. Marketakis, Y. Tzitzikas, R. Guarino, H. Brocks, and F. Engel, “Towards a Long-term Preservation Infrastructure for Earth Science Data,” in *Procs of the 9th Intern. Conf. on Digital Preservation (iPres’2012)*, 2012.
- [38] O. Erling and I. Mikhailov, “SPARQL and Scalable Inference on Demand,” 2009, <http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VOSScalableInference>.
- [39] A. Jackson, “Using automated dependency analysis to generate representation information,” in *Procs of the 8th International Conference on Preservation of Digital Objects (iPres’2011)*, 2011.
- [40] O. Hartig, “Querying trust in rdf data with tsparql,” in *Procs of the 6th European Semantic Web Conference, (ESWC’2009)*, Heraklion, Crete, Greece.
- [41] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.
- [42] P. Bertolazzi, R. F. Cohen, G. D. Battista, R. Tamassia, and I. G. Tollis, “How to draw a series-parallel digraph,” *International Journal of Computational Geometry & Applications*, vol. 4, no. 4, pp. 385–402, 1994.
- [43] C. Meghini, Y. Tzitzikas, and N. Spyrtatos, “Abduction for accessing information sources,” *Fundam. Inform.*, vol. 83, no. 4, pp. 355–387, 2008.
- [44] G. M. Sacco and Y. T. (Editors), *Dynamic Taxonomies and Faceted Search: Theory, Practice and Experience*. Springer, 2009, ISBN = 978-3-642-02358-3.
- [45] M. Belguidoum and F. Dagnat, “Dependency Management in Software Component Deployment,” *Electronic Notes in Theoretical Computer Science*, vol. 182, pp. 17–32, 2007.
- [46] H. Christiansen and V. Dahl, “Assumptions and abduction in Prolog,” in *3rd International Workshop on Multiparadigm Constraint Programming Languages, MultiCPL*, vol. 4. Citeseer, 2004.
- [47] L. Console, D. Dupre, and P. Torasso, “On the relationship between abduction and deduction,” *Journal of Logic and Computation*, vol. 1, no. 5, p. 661, 1991.
- [48] T. Eiter and G. Gottlob, “The complexity of logic-based abduction,” *Journal of the ACM (JACM)*, vol. 42, no. 1, pp. 3–42, 1995.

- [49] X. Franch and N. Maiden, “Modeling Component Dependencies to Inform their Selection,” *2nd International Conference on COTS-Based Software Systems*, Springer, 2003.
- [50] A. Kakas, R. Kowalski, and F. Toni, “The Role of Abduction in Logic Programming,” *Handbook of Logic in Artificial Intelligence and Logic Programming: Logic programming*, p. 235, 1998.
- [51] S. Ross, “Digital preservation, archival science and methodological foundations for digital libraries,” *New Rev. Inf. Netw.*, vol. 17, no. 1, pp. 43–68, May 2012. [Online]. Available: <http://dx.doi.org/10.1080/13614576.2012.679446>
- [52] R. Kowalski and M. Sergot, “A logic-based calculus of events,” *New Gen. Comput.*, vol. 4, no. 1, pp. 67–95, Jan. 1986. [Online]. Available: <http://dx.doi.org/10.1007/BF03037383>

Appendix A

A.1 Epimenides: RDF Schema

```
<?xml version="1.0"?>

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.ics.forth.gr/isl/epimenides">

  <!--Core Vocabulary-->
  <rdfs:Class rdf:ID="Module">
    <rdfs:label xml:lang="en">Module Type</rdfs:label>
    <rdfs:comment>
      The modules that a user holds.
    </rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:ID="WinOS">
    <rdfs:subClassOf rdf:resource="#Module"/>
    <rdfs:comment>A software module.</rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:ID="Task">
    <rdfs:comment>The defined tasks.</rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:ID="Rules">
    <rdfs:comment>A rule determines when a task can
      be performed</rdfs:comment>
  </rdfs:Class>

  <rdf:Property rdf:ID="hasPerformabilityName">
```

```

<rdfs:comment>The unary predicate that denotes
the performability of the task e.g. Run
</rdfs:comment>
<rdfs:range rdf:resource=
  "http://www.w3.org/2000/01/rdf-schema#Literal"/>
<rdfs:domain rdf:resource="#Task"/>
</rdf:Property>
<rdf:Property rdf:ID="hasDepName">
  <rdfs:comment>The binary predicate
  that denotes the dependencies of
  task e.g. runnable
  </rdfs:comment>
  <rdfs:range rdf:resource=
    "http://www.w3.org/2000/01/rdf-schema#Literal"/>
  <rdfs:domain rdf:resource="#Task"/>
</rdf:Property>
<rdf:Property rdf:ID="hasName">
  <rdfs:comment>
    The name of the task e.g. Runnability
  </rdfs:comment>
  <rdfs:range rdf:resource=
    "http://www.w3.org/2000/01/rdf-schema#Literal"/>
  <rdfs:domain rdf:resource="#Task"/>
</rdf:Property>
<rdf:Property rdf:ID="hasRules">
  <rdfs:comment>
    The dependencies of a Task
  </rdfs:comment>
  <rdfs:range rdf:resource="#Rules"/>
  <rdfs:domain rdf:resource="#Task"/>
</rdf:Property>
<rdf:Property rdf:ID="impliesPerformabilityOf">
  <rdfs:comment>
    Is used to express Task-type hierarchies.
    e.g. if I can edit a file then I can also
    read it
  </rdfs:comment>
  <rdfs:range rdf:resource="#Task"/>
  <rdfs:domain rdf:resource="#Task"/>
</rdf:Property>
<rdf:Property rdf:ID="appliedIn">
  <rdfs:comment>The tasks that usually make
  sense to apply to a Module Type</rdfs:comment>
  <rdfs:range rdf:resource="#Module"/>

```

```

    <rdfs:domain rdf:resource="#Task"/>
  </rdf:Property>

  <rdf:Property rdf:ID="asSPARQLRule">
    <rdfs:comment>
      Rules expressed as SPARQL queries
    </rdfs:comment>
    <rdfs:range rdf:resource=
      "http://www.w3.org/2000/01/rdf-schema#Literal"/>
    <rdfs:domain rdf:resource="#Rules"/>
  </rdf:Property>
  <!--End of Core Vocabulary-->

  <!--Mime Types Vocabulary-->
  <rdfs:Class rdf:ID="MIMEType">
    <rdfs:subClassOf rdf:resource="#Module"/>
    <rdfs:comment>
      Basic Module types(Mime) are expressed
      as a subclassOf hierarchy
    </rdfs:comment>
  </rdfs:Class>
  <rdfs:Class rdf:ID="video">
    <rdfs:subClassOf rdf:resource="#MIMEType"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="drawing">
    <rdfs:subClassOf rdf:resource="#MIMEType"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="music">
    <rdfs:subClassOf rdf:resource="#MIMEType"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="application">
    <rdfs:subClassOf rdf:resource="#MIMEType"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="model">
    <rdfs:subClassOf rdf:resource="#MIMEType"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="chemical">
    <rdfs:subClassOf rdf:resource="#MIMEType"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="xgl">
    <rdfs:subClassOf rdf:resource="#MIMEType"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="www">
    <rdfs:subClassOf rdf:resource="#MIMEType"/>

```

```

</rdfs:Class>
<rdfs:Class rdf:ID="x-conference">
    <rdfs:subClassOf rdf:resource="#MIMEType" />
</rdfs:Class>
<rdfs:Class rdf:ID="multipart">
    <rdfs:subClassOf rdf:resource="#MIMEType" />
</rdfs:Class>
<rdfs:Class rdf:ID="i-world">
    <rdfs:subClassOf rdf:resource="#MIMEType" />
</rdfs:Class>
<rdfs:Class rdf:ID="message">
    <rdfs:subClassOf rdf:resource="#MIMEType" />
</rdfs:Class>
<rdfs:Class rdf:ID="audio">
    <rdfs:subClassOf rdf:resource="#MIMEType" />
</rdfs:Class>
<rdfs:Class rdf:ID="x-music">
    <rdfs:subClassOf rdf:resource="#MIMEType" />
</rdfs:Class>
<rdfs:Class rdf:ID="paleovu">
    <rdfs:subClassOf rdf:resource="#MIMEType" />
</rdfs:Class>
<rdfs:Class rdf:ID="image">
    <rdfs:subClassOf rdf:resource="#MIMEType" />
</rdfs:Class>
<rdfs:Class rdf:ID="x-world">
    <rdfs:subClassOf rdf:resource="#MIMEType" />
</rdfs:Class>
<rdfs:Class rdf:ID="text">
    <rdfs:subClassOf rdf:resource="#MIMEType" />
</rdfs:Class>
<rdfs:Class rdf:ID="windows">
    <rdfs:subClassOf rdf:resource="#MIMEType" />
</rdfs:Class>
<!--end of Mime Types-->

<rdf:Property rdf:ID="hasExtension">
    <rdfs:comment>Common extension of
        each mime type</rdfs:comment>
    <rdfs:range rdf:resource=
        "http://www.w3.org/2000/01/rdf-schema#Literal" />
    <rdfs:domain rdf:resource="#windows" />
    <rdfs:domain rdf:resource="#text" />

```

```
<rdfs:domain rdf:resource="#x-world"/>
<rdfs:domain rdf:resource="#image"/>
<rdfs:domain rdf:resource="#paleovu"/>
<rdfs:domain rdf:resource="#x-music"/>
<rdfs:domain rdf:resource="#audio"/>
<rdfs:domain rdf:resource="#message"/>
<rdfs:domain rdf:resource="#multipart"/>
<rdfs:domain rdf:resource="#x-conference"/>
<rdfs:domain rdf:resource="#www"/>
<rdfs:domain rdf:resource="#xgl"/>
<rdfs:domain rdf:resource="#chemical"/>
<rdfs:domain rdf:resource="#application"/>
<rdfs:domain rdf:resource="#model"/>
<rdfs:domain rdf:resource="#music"/>
<rdfs:domain rdf:resource="#drawing"/>
<rdfs:domain rdf:resource="#video"/>
</rdf:Property>

</rdf:RDF>
```