Provenance: From Long-Term Preservation to Query Federation and Grid Reasoning

Herkomstinformatie: Van Langetermijnpreservatie tot Queryfederatie en Netwerkgebaseerd Redeneren

Sam Coppens

Promotoren: prof. dr. ir. E. Mannens; prof. dr. ir. R. Van de Walle

Proefschrift ingediend tot het behalen van de graad van
Doctor in de Ingenieurswetenschappen: Computerwetenschappen

Vakgroep Elektronica en Informatiesystemen
Voorzitter: prof. dr. ir. R. Van de Walle
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2014-2015

Acknowledgements

Writing a PhD dissertation is one of the most difficult things I have ever done. For several years, I have worked hard to achieve this PhD title. And it is not finished yet. At this moment of writing, a few hours before my deadline of submission, I am still struggling to put my thoughts on paper. There is no magic trick, no routine or established work process for publishing research. It is exactly like Thomas Edison said:

> Genius is one percent inspiration, ninety-nine percent perspiration.

I do not want to compare myself to a genius, by far not, but as PhD student, I like to pretend I am. Everybody talks about the inspiration and we all hide the perspiration. Why? Because a true genius is 100 percent inspiration, without perspiration, right? Wrong! Let me get this straight: regardless of how smart you are, it will always be ninety-nine percent perspiration. And this is even a best case scenario. The case where you belong to the happy few, who have enough inspiration to spend one percent of their time on it.

Here am I, talking about the hard work I did to finish this PhD dissertation. But hard work is relative. It is all about perception. Indeed, numerous nights, I have spent doing research, writing papers and articles, preparing presentations. The radio was my best friend. It never let me down. It always reminded me I was not the only one in the world who was awake, working. Another friend who never let me down: coffee. My comfort during those nights, during the following hard mornings. Yet, 211 pages as a result of seven years of (hard) work. That is an average of 41 words a day. 3 sentences a day! And still I have the perception I worked hard for this. A wise man once told me: Working at a research lab is like working at a sheltered workshop, but for intellectuals. There might be some truth in there.

I must admit: I did work hard. But undoubtedly, there must be some inefficiency. I blame peer review for this. A normal peer review process consists of first a major revision, then a minor revision, and finally preparing your print-ready version. This means that before you publish 41 words, you must have rewritten them already three times. And then you are lucky, because during every review stage, you might get a reject. As you can see, doing a PhD can be boring, rewriting the same stuff over and over again. It can be frustrating, because who likes to chew a gum that has been chewed already three times? Still, if you want that publication, you will chew that gum, again... At the same time, that frustration is key for publication. Publication is often a test of strength with your peer reviewers: or you give up, or they give up. If they give up, bingo! Accepted! And frustration can make the difference in this test of strength. Frustration is also very important for writing good reviews. Let me confess: my best reviews were even driven by frustration. Because I wanted to reject a paper, but needed to give a good justification for my rejection. And rejecting a paper, for the sole reason you once wrote a better paper which was rejected, is still not considered a valid reason for rejection...

But why do we want that publication? Of course, because we need it for our PhD, but there is also another reason: we like to travel across the world for the sole purpose of spamming about our work, and networking. Spamming and networking are other key concepts to a successful PhD. My best cited article was written by my network. My second best cited article was cited by my network. My third best cited article was cited only by myself. As I said: spamming and networking, they make the difference.

Now, I do not want to discourage anyone to do a PhD. I recommend it even. It is a very inspiring and enriching experience, but it can only succeed with the right guidance. I would like to express my special appreciation and thanks to my advisor Professor Dr. Ir. Erik Mannens and my supervisor Professor Dr. Ir. Rik van de Walle. Both of you have been a tremendous mentor for me. I would like to thank you for giving me this experience, and encouraging me all those years. I learned a lot from both of you. Your advice on both research as well as on my career have been priceless. A special thanks goes to Erik, who was always pushing me to do this PhD, up to the end. Without you, I would never have finished it. I would also like to thank all my jury members for serving as my committee members for letting my

defense be an enjoyable moment, and for your brilliant comments and suggestions, making my dissertation better. I would also like to thank all my colleagues from MMLab. I had a great time with you, not only working together, but also outside work. Doing conference together left me with remarkable memories.

A special thanks to my family. Words cannot express how grateful I am to my mother, father, and sister for all of the sacrifices that you have made on my behalf. You were always there for me. Without you, I would never have had this chance. I was not always the most easy son and student, but your unconditional support made me the man I am today. Last but not least, I would like to express my appreciation for my wife Jamilia, and our twins, Seth and Finn. You were my motivation, my driving force for this PhD. Jamilia, you supported me always during lots of sleepless nights and you were always there for me. Your support was needed to bring this to a successful end and this PhD can be written also on your name. Seth and Finn, this is also for you guys. You will not realise it for the moment, but both you boys were a huge motivation.

This PhD is dedicated to my father, Ward. All my life, I looked up to you. You were my big example, my inspiration. You taught me the love for technology, from a very young age. We were always discussing the latest evolutions. You were sending me constantly interesting articles on computer science. You paved my way. During my whole life, you stood by me. You supported every decision I have ever made. I could lean on you. Literally, up to your last day, you were motivating me for this PhD. I will never forget this. I will never forget you. You were my biggest fan, and I am your biggest fan. This is for you!

Sam Coppens
Date, 2015

Summary

The content on the Web is growing exponential and the management of all the data is becoming problematic. Semantic Web technologies will overcome this huge problem by letting machines act on the data and actually manage the data. The Semantic Web is an extension of the current Web, with the primary goal of making all the information on the Web machine-interpretable. Together with the rise of the Semantic Web, provenance information has gained a lot of interest. It can tell you who did what with which piece of information to get a history track of the information. On the Semantic Web, this information is machine-understandable, such that even machine agents can perform a trust evaluation of the offered information. In this dissertation, we focus on all aspects of provenance information within the Semantic Web: representing it in a machine-readable form, generating it automatically, disseminating it so machines can easily discover it, and consuming the provenance information.

The information on the Semantic Web is described using Resource Description Data (RDF) to form a huge knowledge graph, where all the information is linked to each other. This allows to easily reuse information from other data providers, as reuse becomes only linking your information graph to the other information graph. As information is being combined with other pieces of information on the Web, the provenance of the information becomes very important. It will tell where the information came from, who created the information, who manipulated the information, which processes were involved in creating and adapting the information, etc.

In the future, this need for provenance information will grow. At this moment, the Semantic Web, and the Linked Open Data cloud in particular, remains a read-only Web. In the near future, this Semantic Web will evolve into a read-write Web, with a uniform protocol to not

only read the data, but also write to the data. From then on, machine agents will be able to correct data, or insert newly discovered links. Provenance information will then become a means of managing your data. It will show who did what to your data, so you can authorise the updates or not. At this point, trust evaluation will become even more important than now with the mash-ups of data.

In the first chapter, we introduce provenance information. Provenance is a very generic concept and knows many different definitions, and applications in different domains. Research on provenance can be categorised into four domains: provenance modeling, provenance generation, provenance publication, and finally, provenance consumption. Despite all the different uses and views of provenance, a general core data model can be presented to describe provenance information. At its core, provenance describes the use and production of entities by activities, which may be influenced in various ways by agents. The first chapter not only introduces provenance information, but also the basic Semantic Web foundations. These foundations are important to understand the remainder of the dissertation. These foundations are explained using the Semantic Web stack, i.e., a set of architectural building blocks or layers for establishing the Semantic Web.

In the second chapter, we focus on the modeling and representation of provenance information in the domain of long-term preservation for cultural heritage information and present PREMIS OWL. PREMIS OWL is a semantic formalisation of the PREMIS 3.0 data dictionary of the Library of Congress (LOC). PREMIS 3.0 are metadata implementation guidelines for digitally archiving information for the long term. PREMIS OWL is in fact a semantic long-term preservation schema. Preservation metadata are a mixture of provenance information, structural metadata, technical information on the digital objects to be preserved and rights information to support the long-term preservation of digital resources. PREMIS OWL is an OWL schema, that can be used as data model for your digital archive. At the same time, it can also be used for dissemination of the preservation metadata as Linked Open Data on the Web. This OWL model allows to easily integrate external information, a feature of data described in RDF, which allows to link to, e.g., technical file format information from an external registry. A last benefit of providing a semantic formalisation of PREMIS is that it can easily be extended to suit your archiveś infrastructure and preservation processes. The PREMIS OWL ontology is closely related to 25 preserva-

tion vocabularies of the Library of Congress. These vocabularies are formalised as SKOS vocabularies. Via these vocabularies, the ontology remains interoperable, while offering sufficient extensibility options to reflect the institutions preservation policies. Every archive can use its own vocabularies to describe their preservation processes as long as these vocabularies are linked to the preservation vocabularies of the LOC.

The third chapter describes a digital long-term preservation infrastructure. The infrastructure has in fact two main goals: preserving the offered content for the long term; and disseminating that content as Linked Open Data, including the preservation metadata. This infrastructure relies on PREMIS OWL to model its preservation metadata. This ontology is supplemented with a semantic Dublin Core layer in our layered semantic metadata model. This enables the archive to deal with the very diverse metadata it has to accept for preservation. This Dublin Core layer is not only used as a common ground for managing the content, but also for the Linked Open Data publication of the records. To guarantee the long-term preservation of the harvested content, our platform has the necessary processes in place to keep the information intact and interpretable, in line with the Open Archival Information System (OAIS) reference model. For building our distributed, digital long-term preservation platform, an integration server was used to orchestrate the different processes, based on SOA technology. The preservation services reside on the archiving server. This server actually has a direct connection to the triple store for the descriptive and preservation metadata and the distributed storage for the referenced multimedia files. The LOD server is used for the dissemination of the content and the provenance information, with a triple store as a storage back-end. This way, our distributed infrastructure is able to satisfy its two main requirements.

The long-term preservation infrastructure produces many different versions of the content to be preserved and their preservation metadata. In the fourth chapter, we introduce a way of publishing this dynamic data, together with all its versions and their provenance information as Linked Open Data, while using persistent identifiers for this dynamic content. The W3C Provenance standard already provides a way of disseminating provenance information. This provenance information actually relates all the different versions of the data. Memento already provides a way for datetime content negotiation. Our framework

actually combines both techniques to publish versioned linked data on the Web. This way, our framework allows to publish the information of a resource as Linked Open Data, including all its previous versions and their provenance information, in a web-accessible manner using persistent identifiers.

Now that we have a way of producing and disseminating provenance information, we can focus on the consumption of provenance information. In the fifth chapter, we show how provenance information can be applied to support query distribution. Prior knowledge is required to federate the queries to the appropriate datasets: each dataset provides its own SPARQL endpoint to query that dataset, and each dataset uses its own vocabulary to describe their information. In this chapter, we propose a federated SPARQL framework to query the global data space, relying on query federation and vocabulary mappings. Our query federation will traverse the owl:sameAs links. Our framework exploits the fact that owl:sameAs relationships are symmetrical and transitive. This means that such linked resources are interchangeable and form the basis for the query rewriting. The provenance of the generated owl:sameAs links is processed to resolve the SPARQL endpoints of linked datasets and their vocabulary mappings to support the query rewriting.

The last two chapters have no direct link to provenance, but are rather extensions to previous chapters. In the sixth chapter, we propose a novel workflow engine. The proposed workflow engine would perfectly replace the workflow engine used in the long-term preservation infrastructure, because it is able to orchestrate and steer its workflows based on external information, e.g., information coming from a technical registry telling which file formats are obsolete and which are not. The workflow engine follows a three-step reasoning process. First, it determines for all the resources in its knowledge base the functionality they need to progress in the workflow. This uses a phase-functionality rule file which binds phases of the workflow to functionalities. During a second phase, the functionalities are mapped to REST service calls using RESTdesc/'s functional descriptions. During the third step, the engine executes the generated service calls and pushes the resource it acted on to the next phase in the workflow using a phase-transition rule file. During each reasoning cycle, external information from Web services or external knowledge bases can be included to steer the workflow composition. At the same time, the split up between func-

tionalities and functional services in the workflow composition benefits the maintenance of the services and the workflows.

The seventh chapter is an extension to the distributed querying framework to become a distributed reasoning framework. In this chapter, we propose an extension to the SPARQL query language to support remote reasoning. Until now, the sparql query language was restricted to simple entailment. Now sparql is being extended with more expressive entailment regimes. This allows to query over inferred, implicit knowledge. However, in this case the sparql endpoint provider decides which inference rules are used for its entailment regimes. In this chapter, we propose an extension to the sparql query language to support remote reasoning, in which the data consumer can define the inference rules. It will supplement the supported entailment regimes of the sparql endpoint provider with an additional reasoning step using the inference rules defined by the data consumer.

The final chapter focuses on the conclusions of this dissertation. The topic of the dissertation is provenance information and provenance has been researched from its different dimensions, i.e., representation, generation, dissemination and consumption. Each chapter in this dissertation tackles one of these areas of research for provenance information. For each of the areas, we show in this final chapter what are the innovations, and contributions.

Samenvatting

De gegevens op het Web groeien exponentieel en het beheer van al deze gegevens wordt steeds problematischer. Het Semantich Web tracht dit probleem op te lossen door de data machine-interpreteerbaar te maken zodanig dat de machines de data kunnen beheren. Met de opkomst van het Semantisch Web werd ook de nood aan herkomstinformatie belangrijker. Deze informatie beschrijft alle processen en actoren die betrokken waren in het creëren van een stukje informatie. Deze dissertatie richt zich op deze herkomstinformatie van data, binnen de context van het Semantisch Web. Dit wil zeggen dat we zullen onderzoeken hoe de herkomstinformatie machine-leesbaar kan worden gerepresenteerd, hoe deze herkomstinformatie automatisch kan worden gegenereerd, hoe de herkomstinformatie moet worden gedissemineerd zodanig dat ze gemakkelijk kan ontdekt worden, ook door machines en tot slot hoe deze herkomstinformatie kan worden aangewend.

Om de informatie machine-interpreteerbaar te maken, beschijft men de informatie in het Semantisch Web door middel van RDF. RDF transformeert de data in feite in een grafe. Hierdoor wordt het ook gemakkelijker om data met elkaar in verband te brengen en data dus te gaan herbruiken. Herbruik wordt dan gewoon grafen met elkaar linken. Het doel hiervan is dat de data op het Web met elkaar in verband wordt gebracht, zoals hyperlinks web paginaś met elkaar verbindt, om op deze manier de informatie op het Web toegankelijk te maken als één grote grafe of kennisbron. Het gevolg hiervan is dat het steeds gemakklijker wordt om informatie op het Web te herbruiken, te manipuleren, te transformeren en te combineren, om zo nieuwe kennis te genereren. De herkomstinformatie van deze nieuwe genereerde kennis is heel belangrijk. Het zal je vertellen waar de verschillende stukjes data vandaan komen, hoe ze gecombineerd werden, welke processen en actoren hierbij betrokken waren, etc. De herkomstinformatie geeft een beeld van de hele geschiedenis van de data. In het Semantisch Web

is deze herkomstinformaties machine-interpreteerbaar, zodanig dat machines kunnen inschatten hoe betrouwbaar de data is op basis van deze herkomstinformatie.

In de toekomst zal de behoefte naar herkomstinformatie alleen maar groeien. Op dit moment is het Semantisch Web voornamelijk een Web waarbij de data enkel in machine-interpreteerbare vorm wordt aangeboden. Dit wil zeggen dat machines enkel leesrechten hebben op deze data. In de toekomst zullen de machines deze data gaan manipuleren en corrigeren en dus schrijfrechten verwerven op deze data. Herkomstinformatie wordt dan een middel om je data te beheren, want deze herkomstinformatie zal je vertellen wie wat met je data heeft gedaan en waarom. Sommige actoren zal je vertrouwen, andere weer niet en hun veranderingen zal je teniet doen. Het inschatten van de betrouwbaarheid van gepubliceerde informatie op het Web zal dus ook steeds belangrijker worden. Een eerste stap richting deze inschatting is het modelleren, genereren en publiceren van de herkomstinformatie, op basis waarvan een vetrouwensinschatting kan gebeuren.

In een eerste hoofdstuk, introduceren we herkomstinformatie. Herkomstinformatie is een zeer generiek concept en kent veel verschillende definities en toepassingen in verschillende domeinen. Onderzoek naar herkomstinformatie kan worden onderverdeeld in vier domeinen: representatie, generatie, publicatie, en ten slotte, consumptie. Ondanks alle verschillende toepassingen en de standpunten omtrent herkomstinformatie, kan een algemeen toepasbaar datamodel voor herkomstinformatie worden vooropgeschoven. Dit data model wordt in het eerste hoofdstuk besproken. Het eerste hoofdstuk introduceert niet alleen herkomstinformatie, maar ook de basistechnologieën van het Semantisch Web. Deze technologische basisblokken zijn belangrijk om de rest van de dissertatie te begrijpen.

In het tweede hoofdstuk, richten we ons op de modellering en representatie van herkomstinformatie op het domein van de langetermijnpreservatie van cultureel erfgoed informatie: PREMIS OWL. PREMIS OWL is een semantische formalisering van de PREMIS 3.0 data woordenboek van de Library of Congress (LOC). PREMIS 3.0 zijn metadata implementatie richtlijnen voor het digitaal archiveren van informatie voor de lange termijn. PREMIS OWL is in feite een semantisch preservatieschema. Preservatiemetadata zijn een verzameling van herkomstinformatie, structurele metadata, technische metadata over de digitale objecten die moeten worden bewaard en rechteninforma-

tie. PREMIS OWL is een schema, dat kan worden gebruikt als data model voor een digitaal archief. Op hetzelfde moment, kan het ook worden gebruikt voor de disseminatie van de preservatiemetadata als Linked Open Data op het Web. Dit OWL model maakt het mogelijk om gemakkelijk externe informatie te integreren in de preservatiemetadata, bijvoorbeeld gegevens van een extern technisch register met informatie over bestandsformaten. Een laatste voordeel van semantisch metadatamodel is dat het gemakkelijk kan worden uitgebreid en aangepast aan de technische architectuur van het archief met zijn bijhorende preservatieprocessen. De PREMIS OWL ontologie is gelinkt aan 25 preservatiewoordenlijsten van de Library of Congress. Deze woordenlijsten worden geformaliseerd als SKOS woordenlijsten. Via deze woordenlijsten blijft de ontologie interoperabel over de grenzen van de archieven heen. Tegelijkertijd kan ieder archief zijn eigen beleid en preservatieprocessen beschrijven door middel van archief-specifieke woordenlijsten te gebruiken in de ontologie, zolang deze woordenlijsten maar gelinkt zijn aan de preservatiewoordenlijsten van het LOC.

Het derde hoofdstuk beschrijft een infrastructuur voor de digitale bewaring op lange termijn van digitale informatie. De infrastructuur heeft in feite twee hoofddoelen: het behoud van de aangeboden informatie voor de lange termijn, en het dissemineren van die informatie als Linked Open Data, waaronder ook de preservatie metadata. Deze infrastructuur is gebaseerd op PREMIS OWL om de preservatiemetadata te modelleren. Deze ontologie wordt aangevuld met een Dublin Core-laag in onze gelaagd semantisch metadatamodel. Dit laat het archief toe informatie te bewaren die beschreven is in zeer uiteenlopende metadatamodellen. De Dublin Core-laag wordt gebruikt als een gemeenschappelijke basis voor het beheer van de inhoud, en voor de publicatie van de records als Linked Open Data. Om de bewaring van de aangeleverde informatie op lange termijn te garanderen, bezit het platform over de nodige processen om de gegevens intact en interpreteerbaar te houden, in lijn met het Open Archival Information System (OAIS) referentiemodel. Voor het bouwen van onze gedistribueerd preservatie platform werd een integratieserver gebruikt om de verschillende preservatieprocessen te orkestreren. De preservatieprocessen zelf zijn ingebouwd in de archiveringsserver. Deze server heeft een directe verbinding met de triplestore voor de beschrijvende metadata en preservatiemetadata en heeft een directe verbinding met de gedistribueerde opslag voor de bestandsgebaseerde opslag van alle te bewaren informatie. De LOD-server wordt gebruikt voor de dissminatie van alle

metadata. Deze server gebruikt de triplestore als backend. Zo kan de gedistribueerde infastructuur aan de twee eisen voldoen: preservatie en disseminatie.

De preservatie-infrastructuur produceert veel verschillende versies van de te bewaren informatie. In het vierde hoofdstuk introduceren we een manier voor het publiceren van deze dynamische gegevens als Linked Open Data. Dit wil zeggen dat we alle versies van de data publiceren als Linked Open Data, alsook de preservatiemetadata die de versies verbinden met elkaar, in combinatie met het gebruik van persistente identifiers voor de bewaarde, geversioneerde data. De W3C Provenance standaard biedt reeds een manier aan voor de disseminatie van herkomstinformatie. De preservatiemetadata vervult in feite de rol van herkomstmetadata. Memento, daarentegen, voorziet een manier voor datetime content negotiatie, wat toelaat verschillende versies onder één persistent identifier te publiceren. Ons raamwerk combineert in feite beide technieken om de verschillende versies van informatie op het web te publiceren als Linked Open Data, tesamen met hun herkomst informatie, gebruik makend van persistente identifiers.

Nu we een manier hebben voor het modelleren, produceren en dissemineren van herkomstinformatie, kunnen we ons richten op de consumptie van herkomstinformatie. In het vijfde hoofdstuk, laten we zien hoe herkomstinformatie kan worden aangewend om query-distributie te ondersteunen. Bij query-distributie kan een client vragen afvuren op een server/eindpunt die de vragen opsplitst in kleinere deelvragen en deze verspreid naar verschillende databronnen. De antwoorden op deze verschillende deelvragen worden dan gecombineerd om een algemeen antwoord voor de binnenkomende vraag te formuleren. Hiervoor is voorkennis nodig. Ten eerste moet men weten waar men de deelvragen naartoe kan sturen (de vraagpunten). Ten tweede moet men voor elk vraagpunt weten in welke woordenschat de data is beschreven zodat de deelvragen zich kunnen aanpassen aan de woordenschat. In dit hoofdstuk stellen we een query-distributie raammwerk voor, gebaseerd op de SPARQL querytaal en queryprotocol. Dit raamwerk maakt gebruik van het feit dat informatie die is gepubliceerd als Linked Open Data links bevatten naar andere gelijkaardige concepten op het Web. Voor het linken van deze gelijkaardige concepten wordt de owl:sameAs relatie gebruikt. Ons query-distributie raamwerk volgt deze owl:sameAs links en gebruikt de herkomstinformatie van deze links om de deelvragen te distribueren. Ons raamwerk verwerkt de her-

komstinformatie van deze links en extraheert hieruit de vraagpunten waar het zijn deelvragen naartoe kan sturen en tevens de woordenschat waarin die data van de vraagpunten is beschreven. Met die informatie kan elke binnenkomende vraag worden opgesplitst in deelvragen, worden de deelvragen aangepast aan de woordenschat van elk vraaagpunt, worden de deelvragen doorgestuurd naar deze vraagpunten en worden uiteindelijk alle antwoorden gecombineerd to een eenduidig antwoord op de binnenkomende vraag. En dit door enkel gebruik te maken van de herkomstinformatie van de gelegde links in de grafe.

De laatste twee hoofdstukken hebben geen directe link naar herkomstinformatie, maar zijn eerder uitbreidingen op vorige hoofdstukken. In het zesde hoofdstuk, stellen we een nieuwe workflow engine voor. De voorgestelde workflow engine zou perfect de workflow engine kunnen vervangen die wordt gebruikt in de preservatie infrastructuur, omdat het in staat is om haar workflows te orkestreren en te sturen op basis van externe informatie, bijvoorbeeld informatie afkomstig van een technisch register met informatie over welke bestandsformaten verouderd zijn en welke niet. De workflow engine volgt een driestappen redeneringsstrategie. Allereerst worden voor alle concepten in haar kennisbbank de functionaliteiten beredeneerd die de concepten nodig hebben om vooruitgang te boeken in de workflow. Het redeneerproces in deze fase wordt bepaald door regels beschreven in een fase-functionaliteitenbestand. De regels bepalen voor elke fase welke functionaliteit ze nodig hebben om die fase te vervolledigen. Tijdens een tweede fase worden de functionaliteiten gemapt op service beschrijvingen. Deze mapping gebeurt tevens door een redeneerstap en de regels hiervoor zijn de service beschrijvingen die met behulp van RESTdesc zijn beschreven. Tijdens de derde stap worden de beredeneerde service calls uitgevoerd. Indien de uitvoering succesvol is, volgt in deze een laatste redeneerstap, nl. een fasetransitie. De fasetransities koppelen in feite alle fases uit een workflow aan elkaar. De fasetransities worden ook met behulp van regels beschreven in het fase-transitiebestand. Na deze laatste fase zijn alle concepten uit de kennisbank één stap opgeschoven in hun workflow. Het hele redeneercyclus wordt nadien herhaald. Tijdens elke redeneercyclus kan externe informatie van webservices of externe kennisbanken worden opgenomen om de workflow compositie sturen. Ook zorgt de opsplitsing tussen functionaliteiten en functionele webservices voor een gemakkelijker beheer van deze services en de workflows waarin ze zijn opgenomen.

Het zevende hoofdstuk is een uitbreiding van het query distributie raamwerk, voorgesteld in het vijfde hoofdstuk. De voorgestelde uitbreiding kan van het query distributie raamwerk een gedistribueerd redeneringsraamwerk maken. Hiervoor stellen we een uitbreiding van de SPARQL querytaal voor om redeneringsprocessen aan de serverkant te ondersteunen. Tot nu toe werd het SPARQL querytaal beperkt tot het meest eenvoudige redeneerproces: grafe matching. SPARQL werd laatst uitgebreid met expressievere redeneringsprocessen. Dit maakt het mogelijk om impliciet afgeleide kennis te bevragen via SPARQL. In dit geval bepaalt de data-aanbieder de inferentieregels die gevolgd moeten worden tijdens het redeneerproces. In dit hoofdstuk stellen we een uitbreiding van de SPARQL querytaal voor waarbij de dataconcument de inferentieregels bepaalt. De inferentieregels van de dataconsument zullen de inferentieregels van de data-aanbieder aanvullen. Op deze manier kan langs de serverkant over de aangeboden data worden geredeneerd en kan de afgeleide kennis onmiddellijk worden bevraagd.

Het laatste hoofdstuk richt zich op de conclusies van deze dissertatie. Het onderwerp van de dissertatie is herkomstinformatie en deze herkomstinformatie wordt onderzocht vanuit al haar verschillende dimensies, dat wil zeggen, representatie, generatie, verspreiding en consumptie. Elk hoofdstuk in dit proefschrift behandelt één van deze onderzoeksgebieden. Voor elk van de gebieden, tonen we in dit laatste hoofdstuk wat de innovaties en bijdragen zijn.

Contents

Chapter 1

Semantic Web and Provenance - An Introduction

In this dissertation, I investigate provenance information for the Semantic Web, an extension of the current Web in which machines can interpret data and entail new information. Provenance information describes the history of a certain Web resource in terms of processes, which act on different entities, and triggered or influenced by agents. Provenance is of interest for many domains, e.g., eScience, information retrieval, workflows, etc. In this dissertation, the provenance information supports long-term preservation and dissemination of cultural heritage information. For this dissertation, I will model provenance information as data model for a digital archive. Next, I will present a framework for the long-term preservation of cultural heritage information, producing provenance information. I will then investigate and propose a way of publishing this provenance information on the Web. Finally, I will show how provenance information published on the Web can be used to support federated querying on the Linked Open Data cloud.

1.1 Introduction

Provenance refers to the sources of information: who did what with which piece of information. It refers to the history of data, in which entities, processes, and agents were involved. The entities, processes, and agents being described in the provenance, and their level of granularity are domain-specific and application-specific.

In scientific research, the main purpose of provenance information is reproducibility of scientific experiments to reproduce new scientific results. Thus, the provenance information here will include information on a sample and the experiments applied to the sample. In a digital archive, on the other hand, provenance information must support and guarantee its long-term preservation. This means provenance information will focus on describing the preservation processes involved in keeping the digital information intact and representable. In a business context, provenance may include information about financial and legal processes (e.g., in contracts) as well as the electronic (e.g., online ordering) and physical (e.g., shipping) processes that have occurred. On the Web, provenance will give information on the creation of the Web resource, but also information on the processes involved for its publication. Next to this, provenance of a Web resource can also include information of its reuse, digital signatures, linking, etc. Provenance of the Web will initially support trust assessment. Or as Tim Berners-Lee stated it in his vision of the Web[2]:

> At the toolbar (menu, whatever) associated with a document there is a button marked "Oh, yeah?". You press it when you lose that feeling of trust. It says to the Web, 'so how do I know I can trust this information?'. The software then goes directly or indirectly back to meta information about the document, which suggests a number of reasons.

The information on the Web is growing exponentially. The management of all this information is becoming problematic. It cannot just be managed by humans anymore. This was the incentive to build an extension to the current Web, i.e., the Semantic Web, a Web where information is not only shared for consumption by humans, but also by machines.

Provenance is one of the elementary building blocks of the Semantic Web. On the Web, information is being reused, copied, and modified a lot by different data providers. Checking if a piece of information is trustworthy or not, is becoming an impossible task on the Web. Provenance information will enable trust assessment on the Web, but at the same time it will support the read-write Semantic Web, as will be discussed in Section 1.5. With the arrival of massive amounts of Semantic Web data (e.g., via the Linked Open Data community) information about the origin of that data, i.e., provenance, becomes an important factor in developing new Semantic Web applications.

Therefore, a crucial enabler of Semantic Web deployment is the explicit representation of provenance information that is accessible to machine agents, not just to humans.

In this dissertation, I will research provenance information, how one can benefit from integrating it into the Semantic Web. Provenance will be approached from the domain of the publication and long-term preservation of cultural heritage information. All facets of provenance will be covered in this dissertation: from modeling provenance information to the generation of provenance information, its dissemination, and finally, the use of the published provenance information.

In this Chapter, I will discuss some important concepts that will return in the other chapters of this dissertation. First, I will give a more detailed view on provenance information and its representation. Then, I will introduce the Semantic Web and its foundations. Afterwards, I have a look at the future of the Semantic Web, as I believe provenance information will become a first class citizen on the Web. Finally, this chapter concludes with an outline for the rest of this dissertation.

1.2 Provenance

Provenance serves many purposes. Various domains are approaching provenance from their own perspective. For this reason, the W3C Provenance Incubator Group has developed a working definition of provenance on the Web:

> ***Provenance Definition***: Provenance of a resource is a record that describes entities and processes involved in producing and delivering or otherwise influencing that resource. Provenance provides a critical foundation for assessing authenticity, enabling trust, and allowing reproducibility. Provenance assertions are a form of contextual metadata and can themselves become important records with their own provenance.

Provenance has already been the subject for many research and development. In workflow systems for eScience, provenance has been studied to support automatic generation of new scientific results. Licensing standards bodies take into account the attribution of information as it is reused in new contexts. Information retrieval communities have

studied how to deal with contradictory and complementary query results. In digital archives, the management of the preserved data relies on the provenance information of the data. Database communities have investigated provenance to support data restorations and to provide low-level provenance at metadata level. Research for provenance covers a broad domain. For provenance on the Web, there are in general four research areas, as shown in Figure 1.1 :

- **Representation**: Here, the modeling of provenance is the focus. Questions here are, e.g., how to describe the objects and the processes, or how to describe versioned data, etc.
- **Generation**: This area is interested in the generation of provenance information and embedding provenance into systems. For instance, how can provenance support database restoration, or how to integrate provenance in workflow engines, etc.
- **Dissemination**: Once you hold provenance information, you still need to publish it and make it accessible on the Web. These are the main questions of this research area.
- **Consumption**: In this area, the research focuses on consuming provenance information for a certain purpose. For instance, how to make different provenance representation interoperable, or how to allow trust assessment, based on provenance information are research topics in this area.

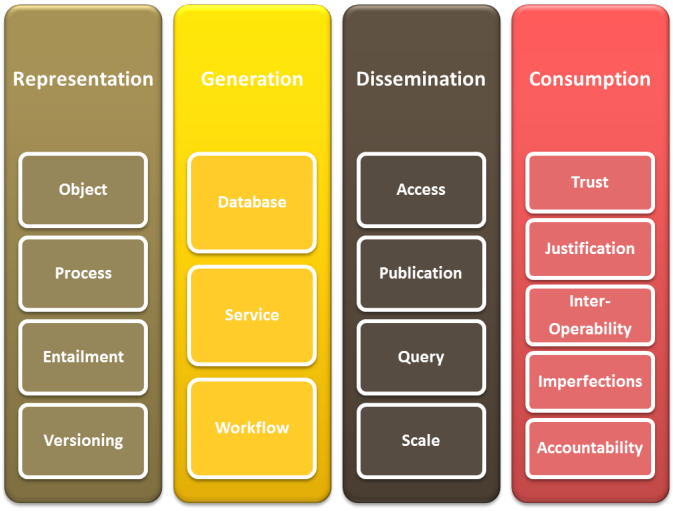

Figure 1.1: Provenance research areas.

Because provenance is used in many different domains, many different data models for provenance have been developed. Each data model was designed to support a certain domain or application. There are very generic provenance models, covering a very broad domain, such as the Open Provenance Model (OPM, [3]). Models to represent provenance for the eScience domain, e.g., the Provenir Ontology [4], provenance for neuromedicine, e.g., the Semantic Web Applications in Neuromedicine Ontology [5], or provenance for Linked Data, e.g., the Provenance Vocabulary [6]. There are also provenance models with a specific goal, such as supporting long-term preservation, e.g., PREMIS [7], describing digital signatures to support trust, e.g., the Semantic Publishing Vocabulary [8], or supporting versioning, e.g., the Changeset Vocabulary [9]. Among the relevant technologies implementing provenance, regardless of their rendering technology, they all share a basic understanding of provenance and a common basis for modeling provenance.

At its core, provenance describes the use and production of entities by activities, which may be influenced in various ways by agents. An **entity** is a physical, digital, conceptual, or other kind of thing with some fixed aspects; entities may be real or imaginary. An **activity** is something that occurs over a period of time and acts upon or with entities; it may include consuming, processing, transforming, modifying, relocating, using, or generating entities. An **agent** is something that bears some form of responsibility for an activity taking place, for the existence of an entity, or for another agent's activity. These three components and their relations constitute the core of most of the provenance models, as depicted by Figure 1.2

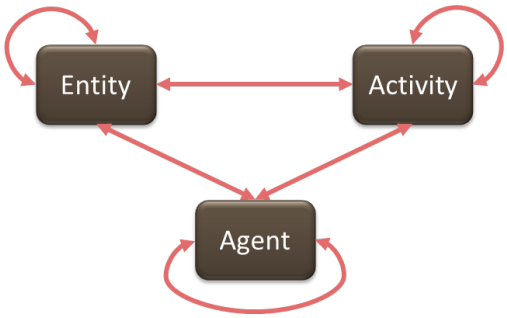

Figure 1.2: Basic components for modeling provenance

Throughout the diversity of provenance models, there are many common threads underpinning the representation, the dissemination and, hence, the use of provenance to enable a new generation of Semantic Web applications that takes provenance into account. For this reason, the W3C has developed a standardised provenance specification in the Provenance Working Group [10]. The aim of this working group is to come up with a common provenance data model, different representations of a provenance data model (e.g., XML, OWL, Dublin Core, etc.) and standardised protocols for discovering, accessing, and querying provenance information on the Web. This specification will give data publishers the tools to publish provenance information on the Web, i.e., a standardised representation for publishing provenance on the Web and a standardised protocol for discovering and accessing provenance information on the Web.

1.3 The Semantic Web

The World Wide Web has been evolving towards the vision of the Semantic Web, an extension of the existing Web through which machines are able to understand the data on the Web and, as a consequence, manage all these data. It promises to infuse the Internet with a combination of metadata, structure, and various technologies so that machine agents can derive meaning from information, make more intelligent choices, and complete tasks with reduced human intervention.
The Semantic Web is an extension on the Web. This means it builds upon the existing Web architecture. Before the Semantic Web, the Web was a web of documents, e.g., Web sites, which were exchanged between two computers (a Web server and a Web browser). Through hyperlinks these documents were interconnected. Enabling technologies for this Web were URIs [11], HTTP [12] and HTML [13]. The first version of the Web, i.e., Web 1.0, was a read-only Web. People could look up information, but could not interact with the information. This was the era of the static Web sites. Later on, this read-only Web became a read-write Web. This stage was characterised by user interaction and collaboration in the form of user generated content, e.g., Youtube[1], and social networks, e.g., Facebook[2] or Twitter[3]. At the moment, the Web is transforming into a Semantic Web, i.e., Web 3.0, where machines

[1] http://www.youtube.com
[2] http://www.facebook.com
[3] http://www.twitter.com

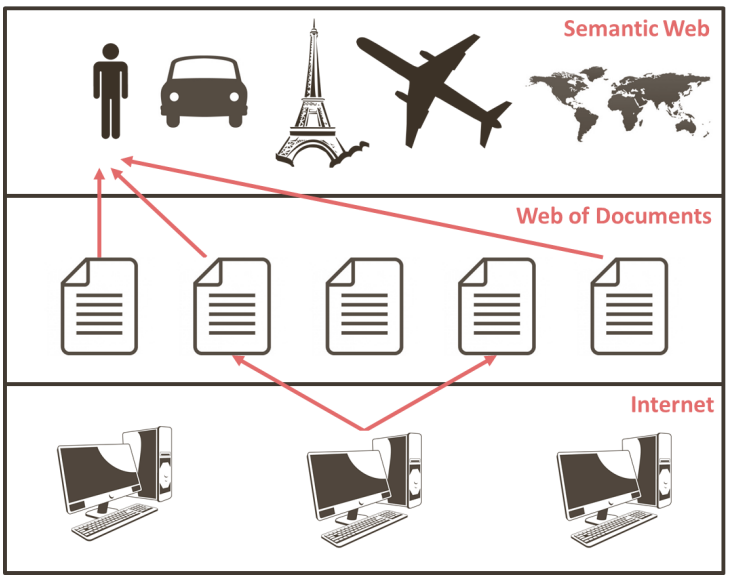

Figure 1.3: Schematic overview of the web evolution

can read and understand the underlying data. In this Semantic Web, we don't describe documents anymore, but things, e.g., a painting of Rubens, Rubens himself, Antwerp, etc. When describing these things, this information can of course come from different sources on the Web, e.g., documents, or services. As such, the Semantic Web becomes a web of interconnected Web resources.

One of the main applications of the Semantic Web is Linked Data (LD, [14]). The goal of Linked Data is to let people share structured data on the Web as easily as they share documents today. It actually refers to a style of publishing and interlinking structured data on the Web. The main recipe for LD is the Resource Description Framework (RDF, [15]). In the Semantic Web, structured data is published as RDF data and RDF links are used to link data from different data sources. Linked Open Data (LOD) is Linked Data that is published openly, freely available. LOD is defined by its 5-star deployment scheme[4] of Tim Berners-Lee. This star scheme for Linked Data indicates how easily the LD can be reused and integrated with other data sources. LOD on the Web creates a giant global graph, where all the published data is freely available and connected to each other, also called the Web of Data. In this Web of Data, clients can easily discover and consume data.

[4] `http://www.w3.org/DesignIssues/LinkedData.html`

1.4 The Semantic Web Foundations

The architecture of the Semantic Web can be illustrated by its technology stack, shown in Figure 1.4. The Semantic Web Stack is an illustration of the hierarchy of languages, where each layer exploits and uses capabilities of the layers below. It shows how technologies that are standardized for Semantic Web are organized to make the Semantic Web possible. It also shows how Semantic Web is an extension (not a replacement) of classical Hypertext Web.

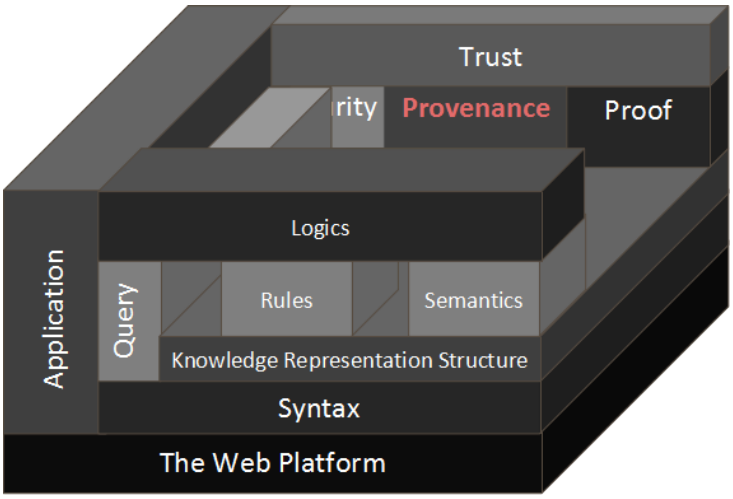

Figure 1.4: Semantic Web Technology Stack

The stack shown in Figure 1.4 is a 3 dimensional visualisation of the layered Semantic Web technology stack, first published by Tim Berners-Lee. Below, I give an overview of some specifications for the layers:

- The Web Platform: URI, HTTP, HTML
- Syntax: N3 [16], Turtle [17], RDFa [18], XML [19], JSON [20]
- Knowledge Representation Structure: RDF
- Semantics: RDFS [21], OWL [22], OWL2 [23]
- Rules: RIF [24]
- Query: SPARQL [25]
- Security: WebID [26]
- Proof: Named Graphs [27]
- Trust: WebID, Web of Trust (WOT)
- Applications: Social Web[5], Linked Open Data

[5] http://www.w3.org/2005/Incubator/socialweb/

1.4.1 The Web Platform: URI

A Uniform Resource Identifier (URI) is a string of characters used to uniquely identify a resource. Such identification enables interaction with representations of the resource over a network (typically the World Wide Web) using specific protocols, e.g., HTTP, or FTP[28]. Anyone can create a URI, and the ownership of them is clearly delegated, so they form an ideal base technology with which to build a global Web on top. The syntax of URIs is carefully governed by the IETF, who published RFC 2396 as the general URI specification. The W3C maintains a list of URI schemes.

URIs can be classified as URLs, as URNs, or as both. A uniform resource name (URN) identifies a resource, e.g., an ISBN number. A uniform resource locator (URL) provides a method for finding this resource, e.g., an HTTP URI. Since 2005, URIs can be used to designate things, not explicitly bound to a document. This allows to identify any thing you want to describe with a URI.

1.4.2 Knowledge Representation Structure: RDF

The underlying structure of Resource Description Framework (RDF) data is a collection of triples. Every triple consists of a subject (S), predicate or property (P) and object (O) respectively. A set of such triples is called an RDF graph.

Figure 1.5: Triple having a resource as its object

Note that the object of a triple can also be a literal instead of a URI. However, literals cannot appear as the subject or predicate of a triple. By convention, a literal is represented using a box instead of an ellipse.

Figure 1.6: Triple having a literal as its object.

All literals have a lexical form being a Unicode string. A literal can also be provided with a language tag. Alternatively, a datatype URI can be provided to a literal, forming a typed literal. If we take the example of Damien Hirst's 'For the Love of God' record and put it in RDF, the record becomes a graph, represented below.

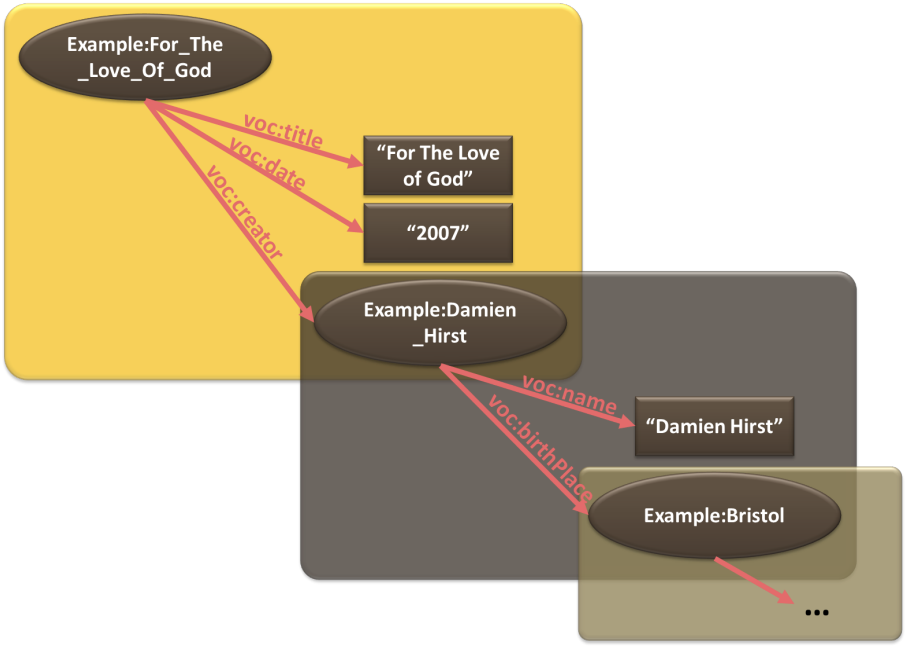

Figure 1.7: Description of artwork of Damien Hirst

1.4.3 Syntax: Serialisation of RDF

RDF transforms data into a data graph. This graph can be serialised for data exchange. As explained earlier, the Semantic Web builds upon the Web of documents, wherein the exchange of documents is the main foundation. When RDF can get serialised into a document, the existing Web architecture can be exploited.

W3C have developed an XML serialization of RDF in the RDF Model and Syntax recommendation. RDF/XML is considered to be the standard interchange format for RDF on the Semantic Web. The example, shown in RDF Fragment 1.1, serialises the example RDF graph of Figure 1.7 to RDF/XML.

RDF Fragment 1.1: Example RDF serialisation in RDF/XML

```
<?xml version="1.0"?>
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:voc ="http://example.org/voc/vocabulary#">
 <rdf:Description rdf:about="http://example.org/resource/
    For_The_Love_Of_God">
   <voc:title>For the Love of God</voc:title>
   <voc:description>For the Love of God is a sculpture by
       artist Damien Hirst produced in 2007. It consists of a
       platinum cast of a human skull encrusted with 8,601
       flawless diamonds, including a pear-shaped pink diamond
       located in the forehead.</voc:description>
   <voc:date>2007-06-01T00:00:00</voc:date>
   <voc:creator rdf:resource="http://example.org/resource/
       Damien_Hirst"/>
 </rdf:Description>
</rdf:RDF>
```

RDF/XML is not the only serialisation of RDF to a document. There exist several other serialisations. For example, Notation3 (N3) is an excellent plain text alternative serialization. N3 is a shorthand, non-XML serialization of Resource Description Framework models, designed with human-readability in mind. N3 is much more compact and readable than an RDF/XML serialisation. RDF Fragment 1.2 shows the RDF graph of Figure 1.7 serialised in N3. N3 has several features that go beyond a serialization for RDF models, such as support for RDF-based rules.

RDF Fragment 1.2: Example RDF serialisation in N3

```
@prefix voc: <http://example.org/voc/vocabulary#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
<http://example.org/resource/For_The_Love_Of_God>
  voc:creator      <http://example.org/resource/Damien_Hirst>
  voc:date         "2007-06-01T00:00:00";
  voc:description  "For the Love of God is a sculpture by \n
                   artist Damien Hirst produced in 2007. It \n
                   consists of a platinum cast of a human \n
                   skull encrusted with 8,601 flawless \n
                   diamonds, including a pear-shaped pink \n
                   diamond located in the forehead.";
  voc:title        "For the Love of God".
```

N-Triples is another way of serialising RDF graphs. It just enumerates all the triples of the RDF graph. It was designed to be a simpler format than Notation 3, and therefore easier for software to parse and generate, but not faster. From now, lots of RDF fragments will be shown. I won't be repeating the namespace declaration for every RDF fragment. The used namespaces throughout this dissertation will be declared once in the text, and will be listed in the namespace list at the beginning of the dissertation.

1.4.4 Semantics: RDFS, OWL, and OWL2

So far, I have introduced RDF, or how data can be modeled as a graph representation using RDF. RDF provides a way to state facts or make assertions about resources, using named properties and values. Users also need to be able to define a vocabulary to use in these statements. RDF properties may be thought of as attributes of resources and in this sense correspond to traditional attribute-value pairs. RDF properties also represent relationships between resources. RDF however, provides no mechanisms for describing these properties, nor does it provide any mechanisms for describing the relationships between these properties and other resources. Thus, RDF does not add semantics to the data.

Semantic models are used to attribute RDF data models with semantics. These models define how your data is described. RDF data can be encoded with semantic metadata using two syntaxes: RDFS and OWL. While RDF is a graph-based model, RDFS (RDF Schema) and OWL are object-oriented. Both RDFS and OWL are W3C specifications.

A. RDFS

RDF's vocabulary description language, RDF Schema, is a semantic extension of RDF. RDFS is defined in the form of an RDF vocabulary. Thus, RDFS descriptions are written in RDF. It is meant to be a simple datatyping model for RDF. RDFS defines classes and properties that may be used to describe classes, properties and other resources. It provides mechanisms for describing groups of related resources and the relationships between these resources. These resources are used to determine characteristics of other resources, such as the domains and ranges of properties, subclasses and subproperties. The namespace of RDFS is <http://www.w3.org/2000/01/rdf-schema#>. The namespace for RDF elements is <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

The most important classes RDFS introduces are rdfs:Resource, rdfs:Class and rdf:Property. Next to these classes, RDFS defines some classes for datatyping, e.g., rdfs:Literal and constructs for representing containers, e.g., rdfs:Container, and RDF statements, e.g., rdf:subject.

Taking back the example of Figure 1.7, one could define a class of artists and a class of artworks. These classes can then be used for datatyping your RDF model.

RDF Fragment 1.3: Example RDFS class declaration in RDFS model

```
voc:Artist  rdf:type       rdfs:Class;
voc:Artwork rdf:type       rdfs:Class;
```

One could state, e.g., that the resource <http://example.org/resource/Damien_Hirst> is an individual belonging to this class. For this, RDFS introduces the property rdf:type. This property denotes to which class a resource belongs. You can add extra triples to your RDF graph to denote which resource belongs to which class. RDF Fragment 1.4 shows which triples are added to our RDF graph shown in Figure 1.7 to datatype the following resources: <http://example.org/resource/Damien_Hirst> and <http://example.org/resource/For_The_Love_Of_God>.

RDF Fragment 1.4: Example RDFS datatyping in RDF

```
<http://example.org/resource/Damien_Hirst>        rdf:type voc:
    Artist.
<http://example.org/resource/For_The_Love_Of_God> rdf:type voc:
    Artwork.
```

For properties, RDFS allows to define their domain and range. The domain must always be a certain class, the range can be a certain class, e.g., voc:Artist, or datatype, e.g., xsd:dateTime. The namespace xsd stands for <http://www.w3.org/2001/XMLSchema#>. In case the range of the property is a datatype, the property is called a datatype property. In case the range is a class, the property is called an object property. In our example RDF model, shown in Figure 1.7, one could define the domain and range of the object property voc:creator, and the datatype property voc:date.

RDF Fragment 1.5: Example RDFS properties

```
voc:date        rdf:type      rdf:Property;
                rdfs:domain   voc:Artwork;
                rdfs:range    xsd:dateTime.
voc:creator     rdf:type      rdf:Property;
                rdfs:domain   voc:Artwork;
                rdfs:range    voc:Artist.
```

Next to this, RDFS also allows to define subclasses and subproperties. In our example, we can define a class voc:ContemporaryArtist as a subclass of voc:Artist and a class voc:ContemporaryArtwork as a subclass of voc:Artwork. Because the property voc:creator is defined to have a voc:Artwork as domain and a voc:Artist as range, this property is still usable for linking a voc:ContemporaryArtwork to a voc:ContemporaryArtist. Thus, instead of defining example:Damien_Hirst and example:For_The_Love_Of_God as resp. voc:Artist and voc:Artwork, one could define them as resp. voc:ContemporaryArtist and voc:ContemporaryArtwork. The RDF example shown in RDF Fragment 1.6 shows how to achieve this subclassing.

RDF Fragment 1.6: Example Subclasses

```
voc:ContemporaryArtwork       rdf:type        rdf:Class;
                              rdfs:subClassOf voc:Artwork.
voc:ContemporaryArtist        rdf:type        rdf:Class;
                              rdfs:subClassOf voc:Artist.
```

If we put everything together, we get our first RDFS model, shown in Appendix A. It is good practice to always add labels and comments to your class definitions and property definitions, because they contribute to the human-readable semantics of the class or property. It is important to make a distinction between this RDFS model and the RDF model as both can be expressed using triples. This allows to join the RDF triples with the RDFS triples to entail some extra information. This is called reasoning, as will be explained in Section C.

B. OWL

OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. This representation of terms and their interrelationships is called an *ontology*. OWL has more facilities for expressing meaning and semantics than RDF, and RDFS, and thus OWL goes beyond these languages in its ability to represent machine-interpretable content on the Web. OWL builds further on RDFS and, as a consequence, is compatible with RDFS, but it allows more expressivity to describe things and to add semantics to descriptions. At the moment, there are two specifications of OWL: OWL1 and OWL2. OWL2 depreciates OWL1, but is compatible with OWL1. To explain the concepts of OWL, I will start from OWL1. Later on, I will expand on OWL2.

OWL1 foresees a number of constructs (classes and properties) to model your data with more expressivity than RDFS. There exist several kinds of constructs. On top of the RDFS constructs (rdfs:Class, rdfs:Property, rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain, rdfs:range, and individuals), OWL1 provides constructs for expressing equality of classes, properties and individuals, characteristics of properties, restrictions on properties, cardinalities, class intersections, and class enumerations.

OWL1 provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users. Each sublanguage has its own restrictions on the introduced constructs of OWL, sacrificing expressiveness for reasoning performance.

OWL Lite has the lowest complexity and supports primarily classification and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. OWL Lite uses only some of the OWL language features and has more limitations on the use of the features than OWL DL or OWL Full. OWL Lite does not use the following features: owl:oneOf, owl:disjointWith, owl:unionOf, owl:complementOf, and owl:hasValue. Furthermore, in OWL Lite classes can only be defined in terms of named superclasses (superclasses cannot be arbitrary expressions), and only certain kinds of class restrictions can be used. E.g., equivalence between classes and subclass relationships between classes are also only allowed between named classes, and not between arbitrary class expressions. The same holds true for owl:intersectionOf. The properties for denoting the cardinality in OWL, are in OWL Lite also restricted to 0 and 1.

Figure 1.8: Constructs introduced by OWL1

OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class). OWL DL is so named due to its correspondence with description logics, a field of research that has studied the logics that form the formal foundation of OWL. Both OWL DL and OWL Full use the same vocabulary (thus, including owl:oneOf, owl:disjointWith, owl:unionOf, owl:complementOf, and owl:hasValue, which OWL LITE does not use) although OWL DL is subject to some restrictions. Roughly, OWL DL requires type separation, .e.g., a class can not also be an individual or property, and a property can not also be an individual or class. This implies that restrictions cannot be applied to the language elements of OWL itself (something that is allowed in OWL Full). Furthermore, OWL DL requires that properties are either ObjectProperties or DatatypeProperties, i.e., DatatypeProperties are relations between instances of classes and RDF literals and XML Schema datatypes, while ObjectProperties are relations between instances of two classes.

OWL Full includes also all OWL language constructs and allows an ontology to augment the meaning of the pre-defined (RDFS or OWL) vocabulary. It is unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL Full.

OWL2 adds new functionality with respect to OWL1. Some of the new features are syntactic sugar (e.g., disjoint union of classes) while others offer new expressivity, including:

- keys
- property chains
- richer datatypes, data ranges
- qualified cardinality restrictions
- asymmetric, reflexive, and disjoint properties
- enhanced annotation capabilities

In addition, some of the restrictions applicable to OWL DL have been relaxed; as a result, the set of RDF Graphs that can be handled by Description Logics reasoners is slightly larger in OWL2. OWL2 also defines three new profiles, instead of OWL Lite, OWL DL, and OWL Full:

OWL2 EL is particularly useful in applications employing ontologies that contain very large numbers of properties and/or classes. This profile captures the expressive power used by many such ontologies and is a subset of OWL 2. Dedicated reasoning algorithms for this profile are available and have been demonstrated to be implementable in a highly scalable way. The EL acronym reflects the profile's basis in the EL family of description logics [EL++], logics that provide only Existential quantification.

OWL2 QL is designed so that the data that is stored in a standard relational database system can be queried through an ontology via a simple rewriting mechanism, i.e., by rewriting the query into an SQL query that is then answered by the RDBMS system, without any changes to the data. Thus, reasoning on these ontologies can be performed by rewriting the incoming queries. The QL acronym reflects the fact that query answering in this profile can be implemented by rewriting queries into a standard relational Query Language.

OWL2 RL is aimed at applications that require scalable reasoning without sacrificing too much expressive power. OWL 2 RL reasoning systems can be implemented using rule-based reasoning engines, hence the RL acronym.

Assume we want to build an ontology for the RDF graph shown in Figure 1.7. With OWL one can express, for instance, that an artwork created after 1945 is a contemporary artwork. An artwork is created by an artist and if the artwork is a contemporary artwork, the artist is a contemporary artist. This can be expressed by the following statements. The full ontology is available in Appendix C. The first thing we do is declaring the classes.

RDF Fragment 1.7: Example OWL class creation

```
:Artwork rdf:type owl:Class .
:ContemporaryArtwork rdf:type owl:Class .
:Artist rdf:type owl:Class .
:ContemporaryArtist rdf:type owl:Class .
```

Next, we define the properties:

RDF Fragment 1.8: Example OWL property declaration

```
:created rdf:type owl:ObjectProperty ;
         rdfs:domain :Artist ;
         rdfs:range :Artwork ;
         owl:inverseOf :creator .

:creator rdf:type owl:ObjectProperty ;
         rdfs:range :Artist ;
         rdfs:domain :Artwork .

:date rdf:type owl:DatatypeProperty ;
      rdfs:domain :Artwork ;
      rdfs:range xsd:dateTime .
```

A contemporary artwork is a subclass of artworks in general. It also inherits its restrictions (exactly one creation date, etc.), but has the extra restriction that its creation date must be after 1945.

RDF Fragment 1.9: ContemporaryArtwork class declaration

```
:ContemporaryArtwork
      rdf:type owl:Class ;
      rdfs:subClassOf :Artwork ,
                      [ rdf:type owl:Restriction ;
                        owl:onProperty :date ;
                        owl:someValuesFrom
                             [ rdf:type rdfs:Datatype ;
                               owl:onDatatype xsd:dateTime ;
                               owl:withRestrictions
                               ( [ xsd:minExclusive "1945-01-01
                                   T00:00:00" ] )
                             ]
                      ] .
```

A contemporary artist is defined by having created at least one contemporary artwork.

RDF Fragment 1.10: ContemporaryArtist class declaration

```
:ContemporaryArtist
      rdf:type owl:Class ;
      rdfs:subClassOf :Artist ,
                      [ rdf:type owl:Restriction ;
                        owl:onProperty :created ;
                        owl:onClass :ContemporaryArtwork ;
                        owl:minQualifiedCardinality "1"^^xsd:
                            nonNegativeInteger
                      ] .
```

C. Reasoning

Reasoning is the process of inferring logical consequences from a set of assertions or axioms. It is used to gain extra knowledge from a graph which is not explicitly stated. This is done by applying some domain knowledge over the data. This domain knowledge can be represented as a set of inference rules, which are commonly specified by an ontology, e.g., RDFS or OWL, or a set of rules, e.g., N3 rules.
Many reasoners use first-order logic to perform reasoning. When inferencing, a set of assertions, often referred to as the A-Box, e.g., the assertions shown in Figure 1.7, is evaluated against a set of inference rules, often referred to as the T-Box, e.g., the ontology designed in Section B. If we take back the example shown in Figure 1.7, we have the following assertions, forming the A-Box:

RDF Fragment 1.11: Example assertions

```
@prefix voc: <http://example.org/voc/vocabulary#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
<http://example.org/resource/For_The_Love_Of_God>
  voc:creator      <http://example.org/resource/Damien_Hirst>
  voc:date         "2007-06-01T00:00:00";
  voc:description  "For the Love of God is a sculpture by \n
                   artist Damien Hirst produced in 2007. It \n
                   consists of a platinum cast of a human \n
                   skull encrusted with 8,601 flawless \n
                   diamonds, including a pear-shaped pink \n
                   diamond located in the forehead.";
  voc:title         "For the Love of God".
```

Let's reason over this A-Box with the ontology designed in Section B. as the T-Box. In this ontology we defined the domain and the range of the property `voc:creator` to be resp. `voc:Artwork`, and `voc:Artist`. Thus, from the assertion

RDF Fragment 1.12: Example assertions

```
<http://example.org/resource/For_The_Love_Of_God> voc:creator <
    http://example.org/resource/Damien_Hirst>
```

a reasoner can infer the following two triples:

RDF Fragment 1.13: Inferred Triples

```
<http://example.org/resource/For_The_Love_Of_God> rdf:type voc:
    Artwork.
<http://example.org/resource/Damien_Hirst> rdf:type voc:Artist.
```

I also defined in the ontology the range of the `xsd:dateTime`, thus the reasoner can infer `"2007-06-01T00:00:00"` as `xsd:dateTime`. Because I defined a `voc:ContemporaryArtwork` to have a creation date younger than 1945, and `voc:ContemporayArtist` to have created at least one `voc :ContemporaryArtwork`, the reasoner will also infer also the following two triples:

RDF Fragment 1.14: Inferred Triples

```
<http://example.org/resource/For_The_Love_Of_God> rdf:type voc:
    ContemporaryArtwork.
<http://example.org/resource/Damien_Hirst> rdf:type voc:
    ContemporaryArtist.
```

1.4.5 Query: SPARQL

To make sure the content can be queried, SPARQL [25] was designed. SPARQL is a query language and data access protocol for the Semantic Web. SPARQL is defined in terms of the W3C's RDF data model and will work for any data source that can be mapped into RDF.

Providing a SPARQL endpoint is very important. It opens up your data and lets other data sources use your data to enrich theirs. SPARQL follows a well-trodden path, offering a simple, reasonably familiar (to SQL users) SELECT query form. The SPARQL query below returns all titles and descriptions of every artwork known by the system. Notice the similarity with SQL, except for the WHERE clauses, which are formulated using RDF.

RDF Fragment 1.15: Example SPARQL query 1

```
PREFIX voc: <http://example.org/voc/vocabulary#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?title ?description
WHERE
{
?artwork a voc:Artwork.
?artwork voc:title ?title.
?artwork voc:description ?description.
}
```

The query can also contain constant values as shown by the query below. In this query I ask for all artworks known by the system of Damien Hirst.

RDF Fragment 1.16: Example SPARQL query 2

```
PREFIX voc: <http://example.org/voc/vocabulary#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?artwork
WHERE
{
?artwork   a   voc:Artwork.
?artwork   voc:creator   <http://example.org/resource/
    Damien_Hirst>.
}
```

SPARQL queries can also handle FILTER expressions on query variables. In the following query, I ask all the artworks of Damien Hirst, made after 2005.

RDF Fragment 1.17: Example SPARQL query 3

```
PREFIX voc: <http://example.org/voc/vocabulary#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?artwork
WHERE
{
?artwork   a   voc:Artwork.
?artwork   voc:creator   <http://example.org/resource/
    Damien_Hirst>.
?artwork voc:date ?date.
FILTER (?date > "2005-01-01T00:00:00" ^^xsd:dateTime)
}
```

SPARQL allows also to do some join operations on the where clauses. To demonstrate this, I will show a query which will ask for all the artworks of Damien Hirst and Jean Michel Basquiat.

RDF Fragment 1.18: Example SPARQL query 4

```
PREFIX voc: <http://example.org/voc/vocabulary#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?artwork
WHERE
{
{
?artwork   a   voc:Artwork.
?artwork   voc:creator   <http://example.org/resource/
    Damien_Hirst>.
}
UNION
{
?artwork   a   voc:Artwork.
?artwork   voc:creator   <http://example.org/resource/Jean-
    Michel_Basquiat>.
}
}
```

1.4.6 Applications: Linked Data

As explained earlier, Linked Data (LD) is one of the main applications of the Semantic Web. LD refers to a way of publishing data, such that clients can easily discover and consume data. The idea behind LD is to create a Web of Data, where all the published data is connected to each other. For this, LOD relies on the following parts of the Semantic Web's technology stack: the Web platform layer (e.g., URI, HTTP), the syntax layer (e.g., turtle, RDF/XML, JSON), knowledge representation structure layer (RDF), the semantics layer (e.g., OWL, RDFS, OWL2), and the query layer (e.g., SPARQL).

LOD stipulates four basic principles:

1. We first have to identify the items of interest in our domain. Those items are the resources, which will be described in the data.
2. Those resources have to be identified by HTTP URIs.
3. Useful information needs to be provided when accessing an HTTP URI.
4. Links have to be made to the outside world, i.e., to connect the data with data from other datasets in the Web of Data. This makes it possible to browse data from a certain server and receive information from another server. In other words, by linking the data with data from other datasets, the Web becomes one huge database, called the Web of Data.

In practice, this means that all your resources should be identified by HTTP URIs. When consumers visit this HTTP URI, the server should provide the user with an appropriate data format. This means HTML for browsers, RDF for other machine agents. This is done by content negotiation [29] or by including RDFa [18] in HTML pages.

The enrichment of the data will actually interlink the data with data from other sources. These enrichments will thus link your data graph to the giant global data graph, as shown in Figure 1.9 [6]. The SPARQL protocol plays an important role in link discovery. In this global data graph data can easily be discovered and consumed. The figure below shows what the global data graph looks like today. Every dot in the graph is a dataset being published as LOD. The interconnections to the LOD datasets are enrichments.

[6] http://lod-cloud.net/

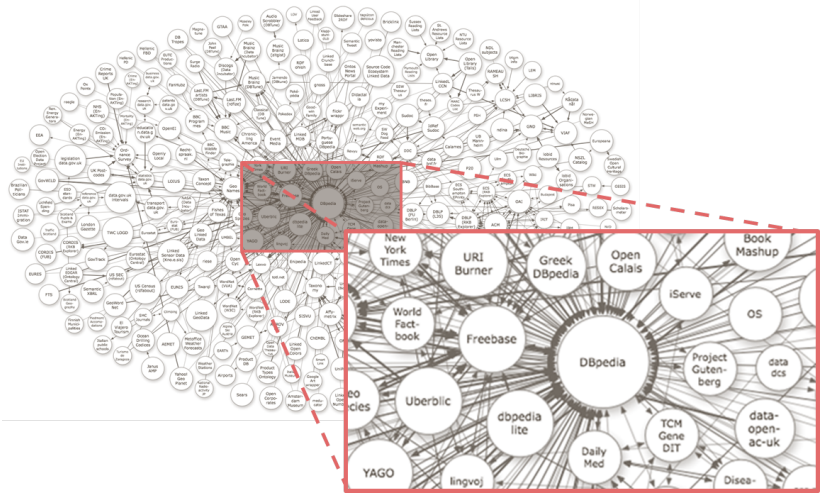

Figure 1.9: Global Data Graph

1.5 Provenance and the Future of the Semantic Web

In Section 1.4, I introduced some of the basic foundations of the Semantic Web. In this section, I have a closer look at the current state and the future of the Semantic Web and the future role of provenance. At the moment, there are indications for an industrial uptake of a number of Semantic Web technologies now that big players, such as Google, IBM and Facebook, have adopted some of these technologies [30].

But it seems that we are still in the phase of publishing existing or new data as structured data on the Web, so that it can be *read* by machines. The current approaches for publishing structured data on the Web usually just provide read-only interfaces to the underlying data. This is also reflected in today's Linked Data applications, which are limited to smart search engines allowing more complex queries, mashups integrating similar data, or classic search engines enhancing their results.

The Web started as a read-only version (i.e., Web 1.0), evolved to a read-write Web where humans can collaborate (i.e., Web 2.0), and is currently a Web powered by semantic technologies so that machines can read and understand the underlying data (i.e., Web 3.0). Given the evolution of the Web, a logical next step towards Web 4.0 seems to be a read-write Semantic Web, where machines can collaborate.

Several proposals exist to enable machine-writable structured data. For instance, Tim Berners-Lee himself proposed update protocols based on WebDAV or SPARQL/Update [31] and implemented them in Tabulator. Lanthaler *et al.* tried to obtain a read-write Web of Data by integrating Web APIs into it (in the form of JSON services) [32]. In 2012, W3C started a standardization activity to create a Linked Data Platform, with as main goal to define a RESTful way to read and write Linked Data [33]. The consequence of this is that published datasets will become more dynamic.

The W3C Provenance Working Group has defined a data model for provenance on the Web and provides guidelines to access information about the provenance of resources on the Web [34]. This specification is a necessary building block before moving towards a specification for the read-write access to Linked Data. Provenance information is not only important for trust assessment of data, it is also necessary for the management of dynamic data, versioning, i.e., revision control, and publishing of this dynamic information. In the future, provenance information will have the potential to become a first class citizen on the Web, however, one that stays behind the scenes, delivering input for many purposes, of which trust, management and revision control are just some examples.

1.5.1 Management

Today, datasets within the Semantic Web are mostly static dumps of information (especially within the Linked Open Data cloud). Moreover, these datasets are not perfect: they contain errors, are not up to date in some cases, and lack links to other datasets. Enabling write access for machines on datasets implies that machines will be able to correct, update, and extend these datasets. Note that, in an ideal world, the human-readable Web is a possible view on the machine-readable Web. This means that, when machines are able to keep the underlying datasets up to date, the human-readable Web will be up to date as well. This way, datasets and data in general on the Web will be driven by machine updates. Indeed, lots of machine agents will, for their specific needs, interpret data and add/correct information where necessary.

Nowadays, schemas (or ontologies) are even more static than datasets. Domain experts carefully discuss each concept and relationship in such a schema. This brings a number of problems such as the gap between a domain expert and an ontology engineer, schema updating is not

trivial, and a lot of schemas are application-specific. In the future, I believe that machines (who will become more and more intelligent and autonomous) will be able to update and correct schemas according to their needs.

Managing these datasets and updated ontologies will become a huge task, certainly when machine agents start correcting your data. Tracking provenance information of all the changes brought to your data will be a requirement for keeping control of your data. Dynamic data will be managed using its provenance information, while for static data some administrative information is enough for its management.

1.5.2 Trust

As explained, currently, most machine-readable data sets are created (scraped, transformed, etc.) from an existing source on the Web. Therefore, today it is in most cases quite trivial to determine the provenance of machine-readable data. With a fully enabled read-write Semantic Web, determining the data provenance will be less trivial. In particular, it should make trust measurements (using whatever metric) transparent for machines (i.e., reproducible). This way, independent parties (i.e., so-called proof checkers or provenance checkers) can judge impartially whether the given data is trustworthy.

As such, provenance information will be crucial for determining trustworthiness of the data. It will deliver the necessary input for the trust measurements and, at the same time, the needed data for the independent proof checking. Provenance is thus crucial for both the a priori, and posteriori trust assessment.

1.5.3 Versioning

Publishing dynamic data on the Web will need extra attention and is different from publishing static datasets. Versioning, or revision control, of the data forms a solution for this. Provenance information will support the versioning of data.

The fact that datasets become more dynamic also has its consequences on reasoning over this data. Most reasoning tasks today are executed on static data sets. With an enabled read-write Semantic Web, structural data will be more and more dynamic. While for some people this might be a reason to use non-monotonic logics to solve future Web inferencing tasks, reasoning on the Web will remain monotonic [35].

In order to keep realizing the latter, changes in the data should be carefully archived, including timestamps. This way, inferencing and proof checking tasks will always be run on a 'snapshot' of the Semantic Web, keeping the reasoning monotonic in a non-monotonic environment. Therefore, I believe that Semantic Web reasoners will be kept monotonic, but will be optimized to deal with timestamp-based (i.e., versioned) data. Here, provenance information will be used for getting the right 'snapshot' of the data.

1.6 Research Questions and Outline

In this chapter, I have introduced provenance information, the Semantic Web, and some of its foundations. I ended with a look at the future of the Semantic Web and the role of provenance information in this Web. In the following chapters of this dissertation, I focus on all aspects of provenance information, i.e., representation, generation, dissemination and consumption, as shown in Figure 1.10

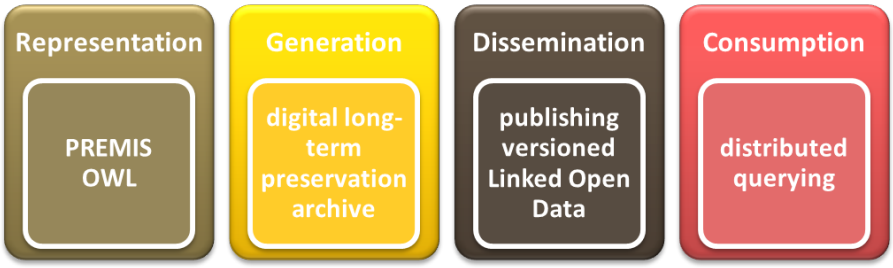

Figure 1.10: Outline of the dissertation

In Chapter 2, I discuss PREMIS OWL, an OWL ontology designed to support long-term preservation. The research question being tackled in this chapter is how we can model a provenance model, which supports at the same time the long-term preservation of the information and the publication of both the preserved information and their provenance information as Linked Open Data on the Web. As I explain, long-term preservation information is in fact provenance information together with rights information and technical information. Thus, PREMIS OWL goes further than merely modeling provenance information and is also able to publish, in addition to provenance information, rights information and technical information as LOD on the Web. An extra requirement of the preservation ontology is that it must be able to overcome interoperability issues of the preservation metadata across

archives. Each archive has its own specific processes in place to guarantee the long-term preservation. This diversity is also reflected on their preservation metadata which makes the metadata not always interoperable across the archives, even when they adhere to the same preservation model.

In Chapter 3, I focus on the generation of provenance information in the form of PREMIS OWL. The main problem being solved in this chapter is how we can build a preservation platform which generates the LOD provenance information automatically. Another research topic of the chapter is how the platform can use the LOD cloud to support its preservation processes. The presented digital archive not only relies on PREMIS OWL for describing its preservation information, but also for managing its objects to be stored, and publishing the generated preservation information as provenance information on the Web.

Such a digital long-term preservation platform keeps track of all versions produced of its objects' information on the Web. In Chapter 4, I will explain how to publish these different versions and provenance information on the Web, such that both the versioned information and their provenance information becomes easily consumable and discoverable. Publishing different versions of the same data on the Web as Linked Open Data poses several problems. Linked Open Data relies on data interlinking. As such, versioning of the data can lead to bad or broken links. In this chapter, I propose a way to publish versioned data on the Web as LOD, that makes the links consistent. For this, I rely on date-time content negotiation. I offer at the same time a publication mechanism for provenance information such that it becomes discoverable on the Web. The provenance information will interlink the different versions of the data and will allow to walk through them.

In Chapter 5, I will describe a distributed querying framework for Linked Open Data. Here, I will show how the published provenance information can be used to support query federation and vocabulary mappings at the same time. This framework will divide an incoming query into subqueries, which will be federated to various SPARQL endpoints. Later, the answers of the subqueries are combined to form an answer for the incoming query. This federated querying framework will actually follow the owl:sameAs links of Linked Open Data for retrieving information on the LOD cloud on the same resource in order to answer an incoming query. The provenance of the owl:sameAs links plays a crucial role in this framework for the federation of the

subqueries and for the vocabulary mappings in these subqueries.

Finally, I end this dissertation with two frameworks that are actually an extension on the discussed work. First, I will propose a semantic workflow engine. The research question here is how we can build a workflow engine that is able to automatically compose and execute workflows. At the same time, the engine needs to fully exploit RESTful services, must be able to be deployed in a distributed network, and must be easily manageable. The workflow engine works in three phases: first, the workflows are composed in terms of functionalities. Later on, these functionalities are fulfilled by REST services, relying on their functional service descriptions. In the last phase, the composed service calls are executed. The workflow engine is powered by semantic reasoning using three rule files, one for each phase. This makes it easily manageable and allows its deployment in the cloud, where each phase can be executed on a different node of the cloud. The proposed workflow engine in this chapter is actually a perfect fit for orchestrating the workflows for a long-term preservation platform, described in Chapter 3.

The last chapter investigates how reasoning tasks of a client can be brought to the server hosting the data. This way, the reasoning tasks are performed where the data resides instead of bringing the data to the reasoning client, which is the case in today's semantic technology stack. For this, I propose an extension to the SPARQL query language and its protocol to support remote reasoning. In this extension to SPARQL, the client is able to define the rules for reasoning over a remote dataset. This extension does not mean SPARQL endpoints need to support the reasoning, but leaves the option open. At the same time, this extension would allow the development of SPARQL clients, which do the heavy reasoning completely client-side or even at a third-party infrastructure. This proposal for remote reasoning support via SPARQL fits current query federation frameworks, as will be described in Chapter 5. These are often based on the SPARQL and could hence be easily extended to become federated, remote reasoning frameworks.

The research performed in this dissertation resulted in the following publications:

Journal papers

- Sam Coppens, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. „PREMIS OWL - A Semantic Long-Term Preservation Model". In: *International Journal on Digital Libraries* (2014)
- Sam Coppens, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. „Self-Sustaining Platforms - Grid Reasoning". In: *Transactions on Autonomous and Adaptive Systems* (2014)
- Erik Mannens, Sam Coppens, Toon De Pessemier, Hendrik Dacquin, Davy Van Deursen, Robbie De Sutter, and Rik Van de Walle. „Automatic news recommendations via aggregated profiling". In: *Multimedia Tools and Applications* (2013), pp. 1–19
- Sam Coppens, Erik Mannens, Toon De Pessemier, Kristof Geebelen, Hendrik Dacquin, Davy Van Deursen, and Rik Van de Walle. „Unifying and targeting cultural activities via events modelling and profiling". In: *Multimedia Tools and Applications* 57.1 (2012), pp. 199–236
- Toon De Pessemier, Sam Coppens, Kristof Geebelen, Chris Vleugels, Stijn Bannier, Erik Mannens, Kris Vanhecke, and Luc Martens. „Collaborative recommendations with content-based filters for cultural activities via a scalable event distribution platform". In: *Multimedia Tools and Applications* 58.1 (2012), pp. 167–213
- Erik Mannens, Ruben Verborgh, Seth Van Hooland, Laurence Hauttekeete, Tom Evens, Sam Coppens, and Rik Van de Walle. „On the Origin of Metadata". In: *Information* 3.4 (2012), pp. 790–808
- Sam Coppens, Jan Haspeslagh, Patrick Hochstenbach, Erik Mannens, Rik Van de Walle, and Inge Van Nieuwerburgh. „Metadatastandaarden, Dublin Core en het gelaagd metadatamodel". eng. In: ed. by Stoffel Debuysere, Dries Moreels, Rik Van de Walle, Inge Van Nieuwerburgh, and Jeroen Walterus. Bewaring en ontsluiting van multimediale data in Vlaanderen : perspectieven op audiovisueel erfgoed in het digitale tijdperk. Lannoo Campus, 2010, pp. 46–62. ISBN: 9789020989441
- Erik Mannens, Sam Coppens, and Rik Van de Walle. „Semantic BRICKS for performing arts archives and disseminiation". In: *IASA JOURNAL* 35 (2010), pp. 40–49
- Sam Coppens, Erik Mannens, and Rik Van de Walle. „Disseminating heritage records as linked open data". In: *International Journal of Virtual Reality* 8.3 (2009), pp. 39–44

Books

- Paul Bastijns, Sam Coppens, Siska Corneillie, Patrick Hochstenbach, Erik Mannens, and Liesbeth Van Melle. „(Meta)datastandaarden voor digitale archieven". dut. In: (2009). Ed. by Rik Van de Walle and Sylvia Van Peteghem, p. 199. URL: http://www.archive.org/details/metadatastandaardenVoorDigitaleArchieven

Book chapters

- Rik Van de Walle, Sam Coppens, and Erik Mannens. „Een gelaagd semantisch metadatamodel voor langetermijnarchivering". In: *BIBLIOTHEEK-EN ARCHIEFGIDS* 84.5 (2009), pp. 17–22
- Sam Coppens, Erik Mannens, and Rik Van de Walle. „Semantic BRICKS for performing arts archives and dissemination". eng. In: ed. by Tom Evens and Dries Moreels. Access to Archives of Performing Arts Multimedia. Vlaams Theater Instituut (VTI), 2009, pp. 121–141. ISBN: 9789074351386
- Stijn Notebaert, Jan De Cock, Sam Coppens, Erik Mannens, Rik Van de Walle, Marc Jacobs, Joeri Barbarien, and Peter Schelkens. „Digital recording of performing arts: formats and conversion". eng. In: ed. by Tom Evens and Dries Moreels. Access to Archives of Performing Arts Multimedia. Vlaams Theater Instituut (VTI), 2009, pp. 95–119. ISBN: 9789074351386

Standardisation activities

W3C

- Luc Moreau, Paolo Missier, Khalid Belhajjame, Reza B'Far, James Cheney, Sam Coppens, Stephen Cresswell, Yolanda Gil, Paul Groth, Graham Klyne, Timothy Lebo, Jim McCusker, Simon Miles, James Myers, Satya Sahoo, and Curt Tilmes. *PROV-DM: The PROV Data Model*. Tech. rep. URL: http://www.w3.org/TR/prov-dm/
- Sam Coppens and Tom De Nies. *PROV-Dictionary: Modeling Provenance for Dictionary Data Structures*. Tech. rep. 2013. URL: http://www.w3.org/TR/prov-dictionary/
- Yolanda Gil, James Cheney, Paul Groth, Olaf Hartig, Simon Miles, Luc Moreau, Paulo Pinheiro Da Silva, Sam Coppens, Daniel Garijo, JM Gomez, et al. „Provenance XG Final Report". In: (Dec. 2010). URL: http://www.w3.org/2005/Incubator/prov/XGR-prov/

- Satya Sahoo, Paul Groth, Olaf Hartig, Simon Miles, Sam Coppens, James Myers, Yolanda Gil, Luc Moreau, Jun Zhao, Michael Panzer, et al. *Provenance vocabulary mappings*. Tech. rep. 2010. URL: http://www.w3.org/2005/Incubator/prov/wiki/Provenance_Vocabulary_Mappings

Library of Congress

- Sam Coppens, Sebastien Peyrard, Rebecca Guenther, Keving Ford, and Tom Creighton. *PREMIS OWL ontology 2.2*. Tech. rep. 2013. URL: http://www.loc.gov/standards/premis/ontology-announcement.html
- Sam Coppens, Sebastien Peyrard, Rebecca Guenther, Keving Ford, and Tom Creighton. *PREMIS OWL ontology 2.1*. Tech. rep. 2011. URL: http://www.loc.gov/standards/premis/owlOntology-announcement.html

Papers in conference proceedings

2013

- Sam Coppens, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. „Querying the Linked Data Graph using owl: sameAs Provenance". In: *Proceedings of the 16th International Conference on Model Driven Engineering Languages and Systems* (2013)
- Sam Coppens, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. „Self-Sustaining Platforms: a semantic workflow engine". In: *Proceedings of the 4th International Workshop on Consuming Linked Data* (2013)
- Miel Vander Sande, Sam Coppens, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. „Adding Time to Linked Data: A Generic Memento proxy through PROV". in: *Poster and Demo Proceedings of the 12th International Semantic Web Conference* (2013)
- Tom De Nies, Sara Magliacane, Ruben Verborgh, Sam Coppens, Paul Groth, Erik Mannens, and Rik Van de Walle. „Git2PROV: Exposing Version Control System Content as W3C PROV". in: *Poster and Demo Proceedings of the 12th International Semantic Web Conference* (2013)
- Sam Coppens, Miel Vander Sande, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. „Remote Reasoning over SPARQL". in: *Proceedings of the 6th Workshop on Linked Data on the Web* (2013)

- Laurens De Vocht, Sam Coppens, Ruben Verborgh, Miel Vander Sande, Erik Mannens, and Rik Van de Walle. „Discovering Meaningful Connections between Resources in the Web of Data". In: *Proceedings of the 6th Workshop on Linked Data on the Web* (2013)
- Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Sam Coppens, Erik Mannens, and Rik Van de Walle. „R&Wbase: Git for triples". In: *Proceedings of the 6th Workshop on Linked Data on the Web* (2013)

2012

- Sam Coppens, Ruben Verborgh, Miel Vander Sande, Davy Van Deursen, Erik Mannens, and Rik Van de Walle. „A truly Read-Write Web for machines as the next-generation Web?" In: *Proceedings of the SW2012 workshop: What will the Semantic Web look like 10years from now* (2012)
- Miel Vander Sande, Sam Coppens, Davy Van Deursen, Erik Mannens, and Rik Van de Walle. „The Terminatorś origins or how the Semantic Web could endanger Humanity". In: ()
- Miel Vander Sande, Ruben Verborgh, Sam Coppens, Tom De Nies, Pedro Debevere, Laurens De Vocht, Pieterjan De Potter, Davy Van Deursen, Erik Mannens, and Rik Van de Walle. „Everything is Connected: Using Linked Data for Multimedia Narration of Connections between Concepts." In: *International Semantic Web Conference (Posters & Demos)*. 2012
- Tom De Nies, Evelien Dh́eer, Sam Coppens, Davy Van Deursen, Erik Mannens, and Rik Van de Walle. „Bringing Newsworthiness into the 21st Century". In: *11th International Semantic Web Conference (ISWC-2012)*. 2012, pp. 106–117
- Ruben Verborgh, Sam Coppens, Thomas Steiner, Joaquim Gabarró Vallés, Davy Van Deursen, and Rik Van de Walle. „Functional descriptions as the bridge between hypermedia apis and the semantic web". In: *Proceedings of the Third International Workshop on RESTful Design*. ACM. 2012, pp. 33–40
- Tom De Nies, Sam Coppens, Davy Van Deursen, Erik Mannens, and Rik Van de Walle. „Automatic discovery of high-level provenance using semantic similarity". In: *Provenance and Annotation of Data and Processes*. Springer Berlin Heidelberg, 2012, pp. 97–110

- Ruben Verborgh, Thomas Steiner, Davy Van Deursen, Sam Coppens, Erik Mannens, Rik Van de Walle, and Joaquim Gabarró Vallés. „RESTdescA FunctionalityCentered Approach to Semantic Service Description and Composition". In: *Proceedings of the Ninth Extended Semantic Web Conference* (2012)
- Erik Mannens, Sam Coppens, Ruben Verborgh, Laurence Hauttekeete, Davy Van Deursen, and Rik Van de Walle. „Automated Trust Estimation in Developing Open News Stories: Combining Memento & Provenance". In: *Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual*. IEEE. 2012, pp. 122–127

2011

- Sam Coppens, Erik Mannens, Davy Van Deursen, Patrick Hochstenbach, Bart Janssens, and Rik Van de Walle. „Publishing provenance information on the web using the Memento datetime content negotiation". In: *WWW2011 workshop on Linked Data on the Web (LDOW 2011)*. Vol. 813. 2011, pp. 6–15
- Sam Coppens, Erik Mannens, Raf Vandesande, and Rik Van de Walle. „Digital long-term preservation, provenance and linked open data". In: *2011 European Library Automation Group conference (ELAG 2011): It's the context, stupid!* 2011
- Sam Coppens, Erik Mannens, and Rik Van de Walle. „What about the provenance of media archives: a tale of time-contextualisation". In: (2011)
- Ruben Verborgh, Thomas Steiner, Davy Van Deursen, Sam Coppens, Erik Mannens, Rik Van de Walle, and Joaquim Gabarró Vallés. „Integrating Data and Services through Functional Semantic Service Descriptions". In: *Proceedings of the W3C Workshop on Data and Services Integration* (2011)
- Toon De Pessemier, Sam Coppens, Erik Mannens, Simon Dooms, Luc Martens, and Kristof Geebelen. „An event distribution platform for recommending cultural activities". In: *7th International Conference on Web Information Systems and Technologies (WEBIST-2011)*. Ghent University, Department of Information technology. 2011, pp. 231–236

2010

- Sam Coppens, Erik Mannens, and Rik Van de Walle. „PREMIS OWL binding to workflow engine for digital long-term preservation". In: (2010)
- Erik Mannens, Sam Coppens, Rik Van de Walle, Laurence Hauttekeete, and Robbie De Sutter. „Cloud-computing approach to sustainable media archives". In: (2010)
- Erik Mannens, Sam Coppens, and Rik Van de Walle. „Network-centric approach to sustainable digital archives". In: (2010)
- Erik Mannens, Sam Coppens, Toon De Pessemier, Hendrik Dacquin, Davy Van Deursen, and Rik Van de Walle. „Automatic news recommendations via profiling". In: *Proceedings of the 3rd international workshop on Automated information extraction in media production*. ACM. 2010, pp. 45–50

2009

- Sam Coppens, Erik Mannens, Tom Evens, Laurence Hauttekeete, and Rik Van de Walle. „Digital long-term preservation using a layered semantic metadata schema of PREMIS 2.0". In: *Cultural heritage on line, Florence, 15th-16th December* (2009)
- Sam Coppens, Erik Mannens, and Rik Van de Walle. „Semantic BRICKS for performing arts archives and dissemination". eng. In: ed. by Tom Evens and Dries Moreels. Access to Archives of Performing Arts Multimedia. Vlaams Theater Instituut (VTI), 2009, pp. 121–141. ISBN: 9789074351386
- Erik Mannens, Sam Coppens, Toon De Pessemier, Kristof Geebelen, Hendrik Dacquin, and Rik Van de Walle. „Unifying and targeting cultural activities via events modelling and profiling". In: *Proceedings of the 1st ACM international workshop on Events in multimedia*. ACM. 2009, pp. 33–40

Chapter 2

PREMIS OWL - A Semantic Long-Term Preservation Model

In this chapter, I present PREMIS OWL. This is a semantic formalisation of the PREMIS 3.0 data dictionary of the Library of Congress (LOC). PREMIS 3.0 are metadata implementation guidelines for digitally archiving information for the long term. Nowadays, the need for digital preservation is growing. A lot of the digital information produced merely a decade ago is in danger of getting lost as technologies are changing and getting obsolete. This also threatens a lot of information from heritage institutions. PREMIS OWL is a semantic long-term preservation schema. Preservation metadata is actually a mixture of provenance information, technical information on the digital objects to be preserved and rights information. PREMIS OWL is an OWL schema, that can be used as data model for your digital archive. At the same time, it can also be used for dissemination of the preservation metadata as Linked Open Data on the Web. The schema is now also approved and managed by the Library of Congress. The PREMIS OWL schema is now published at http://www.loc.gov/premis/rdf/v1#.

2.1 Introduction

The need for digital long-term preservation is growing. A lot of material is still stored on analogue carriers which are degrading rapidely. Not only the material stored on analogue carriers is at danger, also a lot of digital born material. In the digital world, the life cycle of file formats

is very short and many file formats from one or two decades ago aren't supported anymore by today's operating systems and browsers. A digital long-term preservation archive will have all the necessary processes in place to make sure the information remains intact and interpretable.

The heart of such a digital archive is its data model. This data model needs to reflect and support all the preservation processes. Such a data model is called preservation metadata. Preservation metadata exists of provenance information, supplemented with technical information on the digital bitstreams, file formats and representations, rights information and structural information.

In this chapter PREMIS OWL is introduced. PREMIS OWL is a semantic formalisation of the PREMIS 3.0 data dictionary[7]. The PREMIS 3.0 data dictionary defines a conceptual model (i.e. a list of terms) for modeling the preservation information of a digital archive. PREMIS is maintained by the Library of Congress. In fact, this model implements some of the best practices, stipulated by various preservation bodies, e.g, OAIS, the Open Archival Information System, and TDR, Trusted Digital Repositories: Attributes and Responsibilities[81] by the OCLC[1].

First, in Section 2.2, I give an overview of preservation information. Next, in Section 2.3, I introduce more specifically the PREMIS 3.0 Data Dictionary of the Library of Congress. This is a long-term preservation standard, for which a semantic binding is designed. Then, in Section 2.4, the ontology itself is discussed and explained into detail. The chapter ends with a conclusion in Section 2.6.

2.2 Preservation Information

OAIS has defined the responsibilities of a digital archive as follows:

> 'An Open Archival Information System (or OAIS) is an archive, consisting of an organization of people and systems, that has accepted the responsibility to preserve information and make it available for a designated community.'

[1] http://www.oclc.org/

This means that a digital archive is responsible for i) storing the information for the long-term and keeping it in tact and ii) keeping the information interpretable for its designated community, i.e., the primary end-users of the digital archive. These two objectives are subjected to many risks, which a digital archive needs to cover. A first step towards covering these risks, is defining a preservation model that will support tackling these risks. Later on (Chapter 3), the infrastructure can be built to install the necessary processes to preserve the information for the long-term. All the risks inherent to digital long-term preservation can be grouped into five categories spanning a long period of time: interpretation of the file format, bit errors and bugs, file format changes, technology changes and finally, organizational changes.

It is the responsibility of the archive not only to keep the archived information intact, but also interpretable over time. Today, there is a big discrepancy between the **short life-span of file formats** and the need for long-term preservation. File formats, and their different flavours, e.g., TIFF, GeoTIFF, and pyramid TIFF, emerge rapidly. A first risk the archive has to deal with, is the right interpretation of the file format. Further in time, file formats can become obsolete. The archive has then two solutions to present the stored information to the end user: migration or emulation, each needing metadata to support them.

The archive also has to cope with **bit errors, bit rot, and bugs**. Bit rot is the gradual and natural decay of digital information and storage media over time, resulting in eventual unreadability. Bit rot affects different storage formats at different rates depending on the format's durability. Magnetic storage and optical discs are especially subject to varying forms of digital decay. For the time being, masked ROM cartridges appear to be fairly durable while EPROMs are at greater risk. This is why the risk of bit rot comes into play before the risks of file format obsolescence. The archive will need to have processes, and hence also the metadata, in place to correct these errors and to guarantee authenticity of the data. Examples of these are binary metadata, e.g., file format information, fixity information, e.g., MD5 checksums, and digital signatures.

As a next threat the digital preservation platform has to deal with **technology changes**. Technology changes are less frequent than file format obsolescence. Examples of these are for instance *Commodore 64* games, operating system incompatibilities, or even changing technologies regarding information storage, like relational databases, graph

databases, etc. This puts specific demands on the architecture of the archive: it has to organise its data in a platform independent manner. The OAIS reference model, discussed in the next section, provides a high level architecture to deal with this issue.

In the long run, both **institution structures**, terminologies, and the intended audience for your information might change. In practice, this means your descriptive metadata can change, and the metadata format used for it. Other issues at the same level are the rights of an archived object or institution, which can change over time too. To keep the information interpretable, the archive needs: descriptive metadata, for a general description of the object, e.g., MARC; rights metadata, for describing copyright statements, licenses, and possible grants that are given; and context metadata, for describing the relations of the content information to information from external data sources.

When developing a long-term preservation archive, all these described risks have to be taken into account. They have an impact on the architecture used for the archive, the data model used for the archive, and the processes it must have in place for keeping the information meaningful for the end-user. The OAIS reference model is a basis model for long-term archives. It assures platform-independent access to all archived information. In general, it describes the digital archive using three types of packages. Each package aggregates the needed metadata, according to its function, and the digital files accompanying the metadata. These packages are:

- Submission Information Package (SIP): This is the package the archives accept for ingest. It consists of descriptive metadata, digital multimedia files, referenced by the metadata and some additional metadata, e.g., structural metadata or rights metadata.
- Archival Information Package (AIP): This is the package the archives use for preservation. It is in fact the SIP supplemented with preservation information.
- Dissemination Information Package (DIP): This is the package the archives offer for dissemination, e.g., if the rights do not permit to publish the archived digital files accompanying the metadata, this package will only contain metadata.

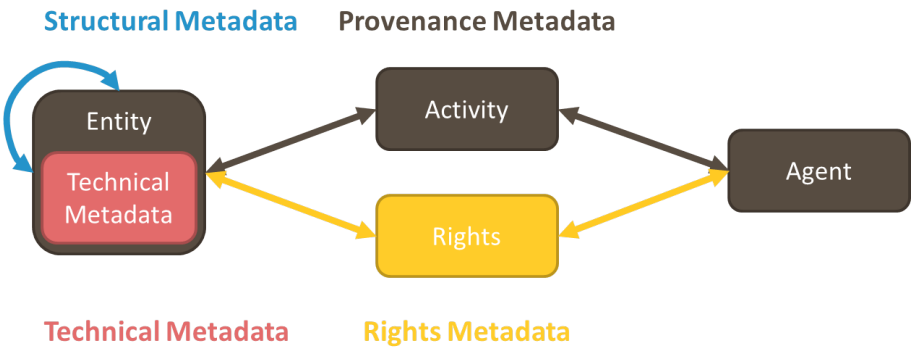

Figure 2.1: Conceptual Model of Preservation Metadata.

The data model for the archive will extend the SIP with preservation information to become an AIP and support the preservation processes of the archive. In general, this preservation information can be seen as an aggregation of four types of metadata:

- Provenance Information: This information will allow to describe the whole history of an object to be stored for the long-term.
- Technical information: The technical metadata will support processes to guarantee the data remain intact and interpretable. One of the processes will be the transcoding of file formats becoming obsolete. Technical information will support these actions.
- Rights metadata: Rights information is needed for the archive to know who can do which action on the stored information.
- Structural information: Objects to be stored can be supplied to the archive as an aggregation of different entities (for instance an HTML page or a book consisting of an ordered number of scanned TIFF images). When preserving these aggregated objects, its structure also needs to be preserved.

If I take back the generic provenance model, defined in Chapter 1, we get the model shown in Figure 2.1. It consists of provenance information, in the form of Entities, Activities and Agents being interlinked. The Entities are actually the objects to be preserved. Their descriptions need to be extended with technical metadata and structural metadata. Finally, the provenance model is also extended with the Rights information to form in its whole a basic preservation model.

2.3 PREMIS 3.0

PREMIS [7] is a preservation standard based on the OAIS reference model. This standard was called the *Data Dictionary for Preservation Metadata: Final Report of the PREMIS working group*[2]. Next to the data dictionary, an XML schema that implements the data dictionary for digital preservation, is also published. This preservation standard is described by a data model, which consists of five semantic units or classes important for digital preservation purposes:

- *Intellectual Entity*: a part of the content that can be considered as an intellectual unit for the management and the description of the content. This can be for example a book, a photo, or a database.
- *Object*: a discrete unit of information in digital form, typically multimedia objects related to the intellectual entity.
- *Event*: An action that has an impact on an object or an agent.
- *Agent*: a person, institution, or software application that is related to an event of an object or is associated to the rights of an object.
- *Rights*: description of one or more rights, permissions of an object or an agent.

Intellectual entities, events, and rights are directly related to an object, whereas an agent can only be related to an object through an event or through rights, as can be seen on Figure 2.2. This way, not only the changes to an object are stored, but the event involved in this change is also described. These relationships offer the necessary tools to properly store the provenance of an archived object. The rights metadata needed for preservation are covered by the rights entity. Technical metadata and structural metadata are encapsulated in the PREMIS data dictionary via the description of the object entity.

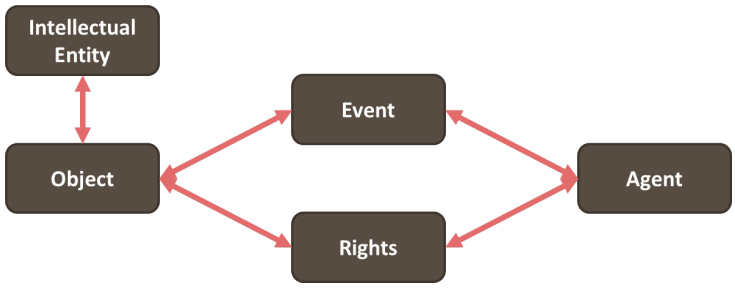

Figure 2.2: Data Model of the Premis 3.0 Data Dictionary.

[2] http://www.loc.gov/standards/premis/v2/premis-2-1.pdf

If I relate this back to the reference model of Figure 2.1, we notice both figures are very similar. The only difference is that the Entities to be preserved in Figure 2.1, are split in PREMIS into Intellectual Entities and Objects. This is because a digital archive stores records or aggregations (collections) of records. Such a record can be seen as a package of descriptive metadata and files that are referenced in the descriptive metadata, e.g. images or videos, as shown in Figure 2.3. The Intellectual Entity of PREMIS is actually the descriptive metadata of the record and the Objects of PREMIS are the referenced files. Now, because the descriptive metadata (Intellectual Enties) can change also over time, they have also an Object equivalent, submitted to Rights and Events. Modeling the Intellectual Entity is not the purpose of PREMIS. PREMIS is concerned in modeling preservation metadata. The Intellectual Entity is descriptive metadata, for which every archive can choose its own appropriate model, e.g., Dublin Core, MARC, EAD, CDWA, ...

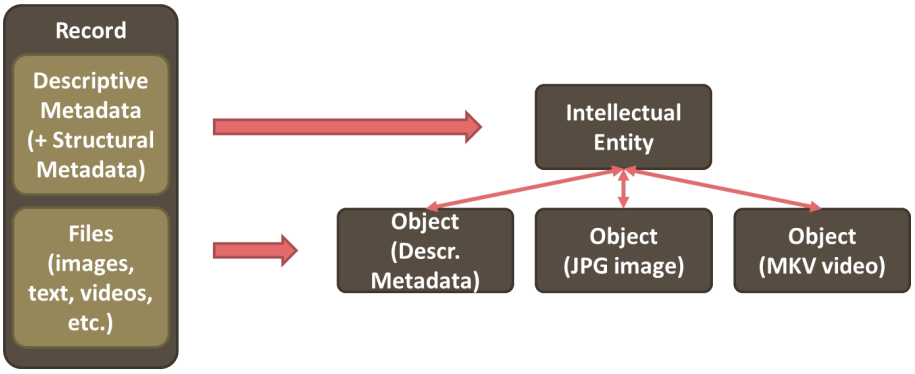

Figure 2.3: Data Model of the Premis 3.0 Data Dictionary.

2.4 PREMIS OWL

The PREMIS OWL ontology is published at <http://id.loc.gov/ontologies/premis.rdf> and its documentation can be found at <http://id.loc.gov/ontologies/premis.html>. The namespace for PREMIS OWL is <http://www.loc.gov/premis/rdf/v1#> and is often shortened to 'premisowl'. For the remainder of this chapter, this namespace will be considered the base URI. It needs to be stressed that PREMIS is a data model for the management of the archive, though on top of that, PREMIS can be used to disseminate

the preservation information for instance as Linked Open Data, albeit it not being its primary concern. When designing the OWL ontology of the PREMIS 3.0 the choice was hence made to stick as closely as possible to the data dictionary of PREMIS 3.0, although it was not always possible or appropriate to do so in OWL. The reason to stick to the data dictionary is that the data dictionary of PREMIS 3.0 was developed by experts in the domain of long-term preservation, and every element has its own clearly defined semantics.

Looking at the data model, one can notice it is dynamically relating the five entities to each other. Until now an XML schema[3] was available that implemented the PREMIS 3.0 data dictionary. Implementing the data dictionary using the Web Ontology Language (OWL, [22]), allows us to relate the entities to each other in a more harmonious way, because RDF is resource based (and, as a consequence, every subject is identified by its URI). Another advantage of using OWL to implement the PREMIS 3.0 data dictionary, is that the relations can be made bidirectional using inverse properties. Using this semantic model of the data dictionary helps to keep the whole archive more consistent and to reuse information as much as possible. At the same time, this OWL model allows to easily integrate external information, a feature of data described in RDF, which allows to link to, e.g., technical file format information from an external registry. A last benefit of providing a semantic formalisation of PREMIS is that it can easily be extended to suit your archive's infrastructure and preservation processes.

The PREMIS OWL ontology is closely related to 24 preservation vocabularies the Library of Congress published at <http://id.loc.gov/preservationdescriptions/>. These vocabularies are formalised as SKOS vocabularies. This way, the ontology remains interoperable, while offering sufficient extensibility options to reflect the institutions preservation policies. For the remainder of this chapter, the namespace of the preservation vocabularies is shortened to 'idlc'.

[3] `http://www.loc.gov/standards/premis/premis.xsd`

In this section, the PREMIS OWL ontology is explained into detail and some design decisions made, are discussed. For describing the PREMIS OWL ontology, I give first an overview of core PREMIS OWL classes and its structural information. Later on, I expand on each of the five PREMIS entities, focusing on the ontology features. PREMIS OWL is a huge ontology, so I won't be able to describe all the formalisation details. For this, I refer to the ontology itself.

2.4.1 PREMIS OWL: Core

For each of the five PREMIS entities, an OWL class was introduced, forming the core of PREMIS OWL. Thus, we have the following classes: *IntellectualEntity*, *Object*, *Event*, *RightsStatement*, and *Agent*. These classes are related to each other using the object properties *hasObject*, *hasEvent*, *hasAgent*, and *hasRightsStatement*. In Figure 2.4, this model is depicted.

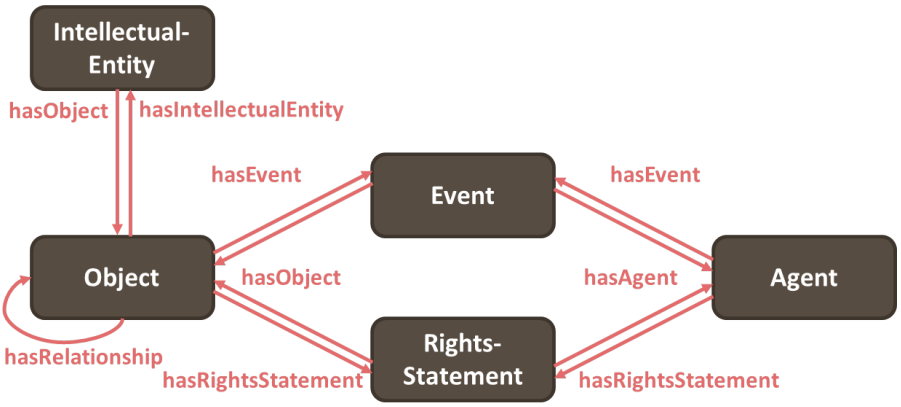

Figure 2.4: Core Classes and Properties of PREMIS OWL.

The relating properties are very general. Sometimes, one needs more specialised relationships to relate the entities, for instance to denote a certain role of the relating entity. For this reason, specific subproperties were created. These subproperties are then further detailed via idlc preservation vocabularies, which are formalised as separate SKOS vocabularies.

An *Object* can have several roles when linked to an event. For this reason, the *hasEventRelatedObject* property is introduced. The idlc:eventRelatedObjectRole vocabulary contains several subproperties of the *hasEventRelatedObject* property. Possible values are *source* and *outcome*.

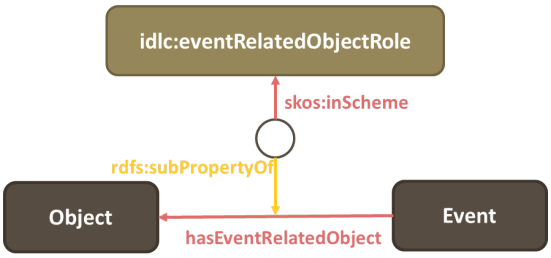

Figure 2.5: Specialised Subproperties for *hasEventRelatedObject*.

The same holds true for an *Agent* related to an *Event*. The property *hasEventRelatedAgent* has been defined as a subproperty of the *hasAgent* property. The idlc:eventRelatedAgentRole vocabulary contains four subproperties to *hasEventRelatedAgent* for denoting an *Agent*'s role in an *Event*, i.e., *authorizer*, *executing program*, *implementer*, and *validator*.

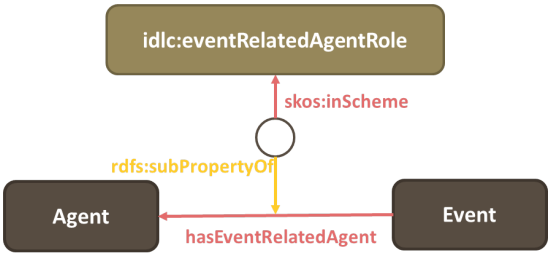

Figure 2.6: Specialised Subproperties for *hasEventRelatedAgent*.

Finally, an *Agent* can also have different roles when linked to a *RightsStatement*. The property *hasRightsRelatedAgent* is introduced as a subproperty to the *hasAgent* property. The idlc:rightsRelatedAgentRole vocabulary contains three subproperties to *hasRightsRelatedAgent* denoting the *Agent*'s role, i.e., *contact*, *grantor*, and *rightsholder*.

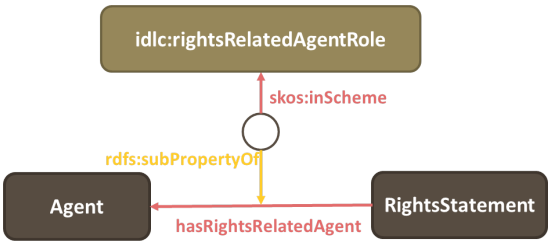

Figure 2.7: Specialised Subproperties for *hasRightsRelatedAgent*.

2.4.2 PREMIS OWL: Structural Information

The *Object* instances can have simple to complext relationships between them. For this PREMIS OWL has the *hasRelationship* property. This is of course too general. The *hasRelationship* has also some subproperties for a finer-grained notion of the relation between two *Objects*. There is a vocabulary for this: idlc:relationshipSubtype. This vocabulary has eight members, e.g., *hasPart*, *hasSource*, or *includes*.

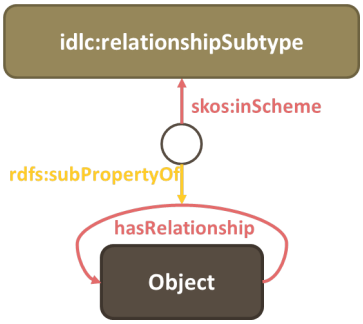

Figure 2.8: Specialised Subproperties for *hasRelationship*.

Sometimes, relationships are more complex and include a certain sequence. For these cases, the class *RelatedObjectIdentification* is introduced. It is related through the same property, i.e., *hasRelationship*, to an object, but at the same time, the class *RelatedObjectIdentification* allows to describe its sequence via the *hasRelatedObjectSequence* property, as depicted in Figure 2.9

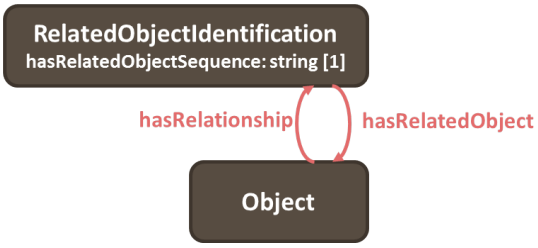

Figure 2.9: Describing Sequences via the *RelatedObjectIdentification* Class.

All the instances of these core classes of PREMIS OWL can have, in addition to a URI, other identifiers. For this, PREMIS OWL foresees an *Identifier* class for describing these identifiers. This class is defined by two mandatory properties, i.e., *hasIdentiferType* and *hasIdentifierValue*. Both properties have a string as range. To link a PREMIS OWL core class instance to the *Identifier* instance, the object property *hasIdentifier* is used.

2.4.3 Object

As explained in Figure 2.1, the *Object* class will contain a lot of technical metadata. This is because of the responsibilities of the digital archive, i.e., keep the information intact, and keep the information interpretable. In order to fulfil these requirements, the archived objects are described carefully with information such that it can check and take the necessary precautions to keep the archived object intact and interpretable. Intact mainly means assuring the integrity of the stored digital information. Interpretable means assuring the stored digital information remains render-able.

The *Object* class has three subclasses: *Bitstream*, *File* and *Representation*, as shown in Figure 2.10. Thus, the objects related to a record to be preserved, can be described at different levels. This is needed, because at every level there are preservation risks involved. On the lowest level

the data inside a file can be described using *Bitstream*. At a higher level, the file itself, of course, can be described using the *File* class. Finally, an object can also be described at representation level. For this, we have the *Representation* class. Typically, a representation is a set of files with some structural metadata relating the files, e.g., a book as a set of ordered TIFF images. Actually the *Object* class has another subclass: *IntellectualEntity*. The *IntellectualEntity* is a class holding the descriptive metadata of a record to be preserved. The descriptive metadata of a record can change over time too, and even the descriptive metadata rights can also be declared. By making the *IntellectualEntity* a subclass of *Object*, one can describe the events related to the descriptive metadata, e.g., reconcilliation, and the rights related to the descriptive metadata.

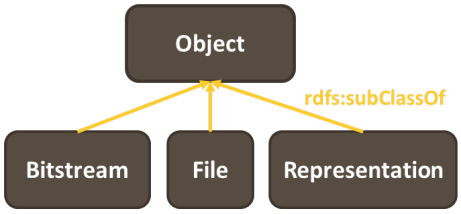

Figure 2.10: Subclasses of the *Object* Class.

Every archive can have different preservation policies. Even within an archive, there can exist several preservation policies. Via the *PreservationLevel* class, as depicted in Figure 2.11, one can assign a certain level of preservation, e.g., some objects may need dissemination online, others don't. This can impact certain transcoding strategies for these objects. Next to the actual preservation level value, one can also describe the rationale behind the preservation level and the role of the preservation level via a member of the preservation vocabulary idlc:preservationLevelRole. Example values of this vocabulary are 'requirement' for denoting what the archive is required to archive, or 'intention' for what is intended to be preserved. The preservation level can only be described for *Representations* and *Files*, because *Bitstreams* are always contained in a file, which has the preservation level attached. Sometimes, an object has certain characteristics that need to be preserved. This can be done by the class *SignificantProperties*. Via this class, these characteristics that need to be preserved, can be described via the *hasSignificantPropertiesValue* and *hasSignificantPropertiesType* properties, as depicted in Figure 2.12

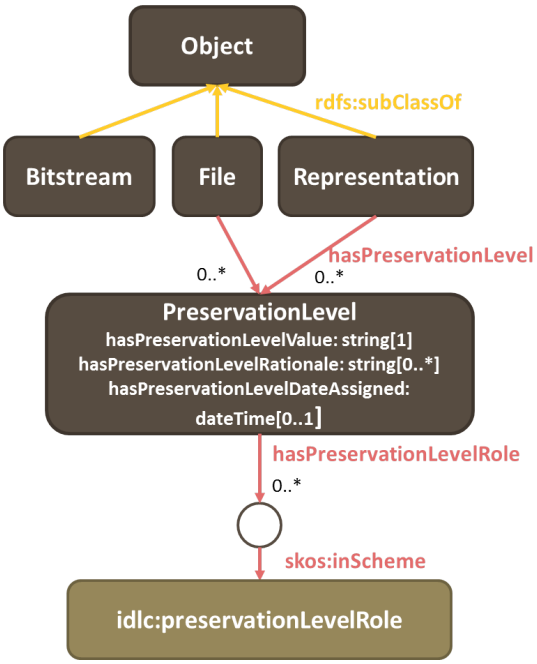

Figure 2.11: Overview of the *PreservationLevel* Class.

Once preserved, instances of *Objects* can be digitally signed to guarantee their authenticity. These digital signatures can be described using the *Signature* class, shown in Figure 2.13. This class lets you describe the signature value, the signature's method and encoding, the validation rules, key information and some of its properties. For the encoding and

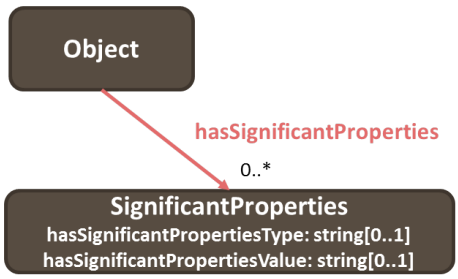

Figure 2.12: Overview of the *SignificantProperties* Class.

the method, the ontology relies on the resp. preservation vocabularies idlc:signatureMethod and idlc:signatureEncoding. Of course, this class is only foreseen for the *Bitstream* and the *File* class, as a *Representation* consists of *Files* and maybe even *Bitstreams*.

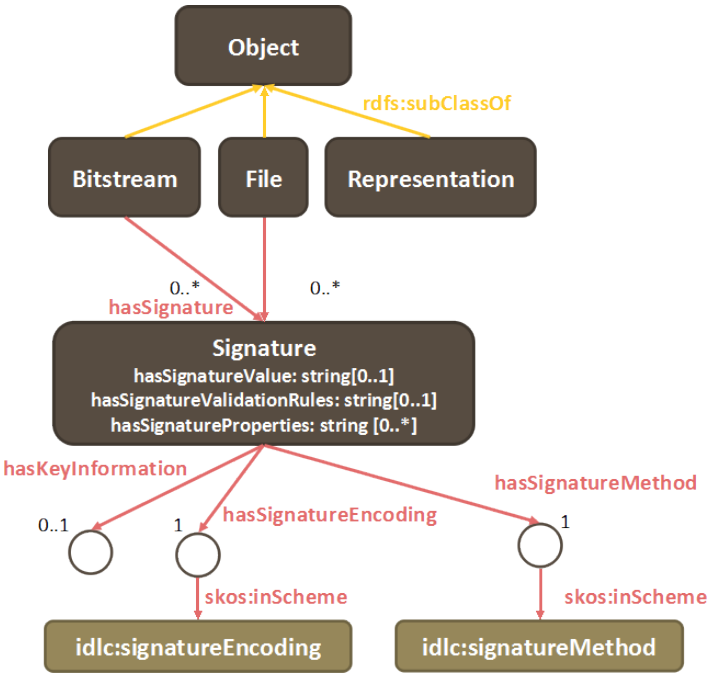

Figure 2.13: Overview of the *Signature* Class.

All the digital objects that need to be preserved, need to be stored, of course. Some archiving policies will prescribe storage on tape, while other may prescribe storage on hard disks. The storage of a preserved object can be described via the *Storage* class, as depicted in Figure 2.14. The *Storage* class makes it possible to describe the storage medium, for which the values are taken from the preservation vocabulary idlc:storageMedium. and the content location. A content location is characterised by a value and a type, taken from the preservation vocabulary idlc:contentLocationType, e.g., 'Handle' or 'URI'. For the same reason signatures can only be described for bitstreams and files, the storage can also only be described for bitstreams and files.

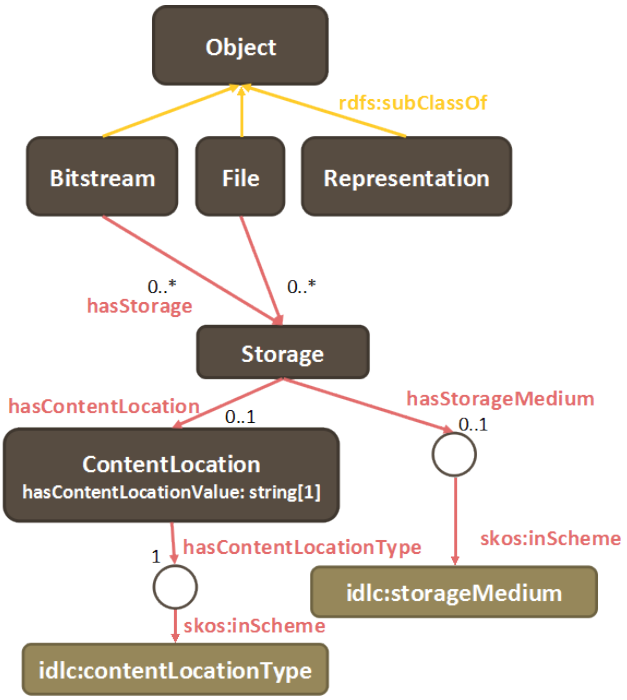

Figure 2.14: Overview of the *Storage* Class.

Of course, the file format information also needs to be described in an *Object* class. This is done through the *ObjectCharacteristics* class. This class aggregates some bit-level information, such as the file format using the *Format* class, the creating application via the *CreatingApplication* class, fixity information through the *Fixity* class, and inhibitor information when encryption is used.

The *Format* class, as shown in Figure 2.15, is equivalent to the *udfrs:AbstractFormat* class and the *pronom:File-Format* class. Thus, these classes are interchangeable. The *Format* class is described either by a format designation, which allows describing the format name and format version, or by an identifier from a format registry. Next to the registry key, the role of the registry can also be described using the preservation vocabulary idlc:formatRegistryRole, e.g., 'specification' or 'validation profile'.
A creating application is further detailed by a name of the creating application and a version. Fixity information is described further by its message digest, its message digest originator, and a message digest algorithm, for which the preservation vocabulary

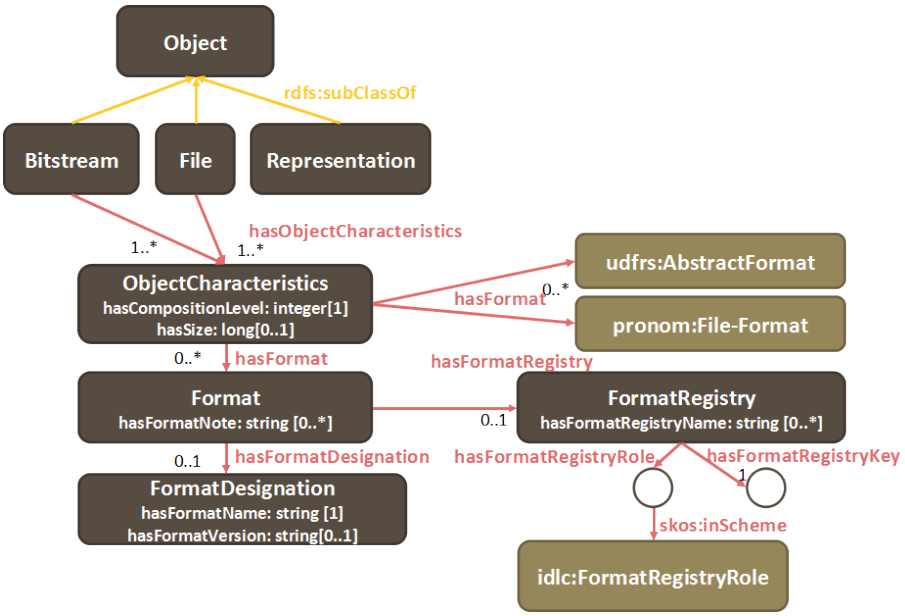

Figure 2.15: Overview of the *Format* Class.

idlc:cryptographicHashFunctions is used. The inhibitors, when encryption is used on an object, are characterised by a key, a target and a type. For the last two, the resp. preservation vocabularies idlc:inhibitorTarget and idlc:inhibitorType are used. An example value from the idlc:inhibitorTarget vocabulary is 'print function', from the idlc:inhibitorType vocabulary is 'PGP'. This information is depicted in Figure 2.16.

Finally, in addition to the file format information, information on the rendering environment needs to be stored. To keep information interpretable, there are basically two options: transcoding or emulation. The file format information will support transcoding, the rendering environment information will support the emulation. For this, the *Environment* class exists, as shown in Figure 2.17. This class basically allows to detail the rendering hardware and the rendering software, via the resp. classes *Hardware* and *Software*. Next to this, the *Environment* class also allows describing some environment characteristics, using the preservation vocabulary idlc:environmentCharacteristic, and the environment purpose, via values from the preservation vocabulary idlc:environmentPurpose.

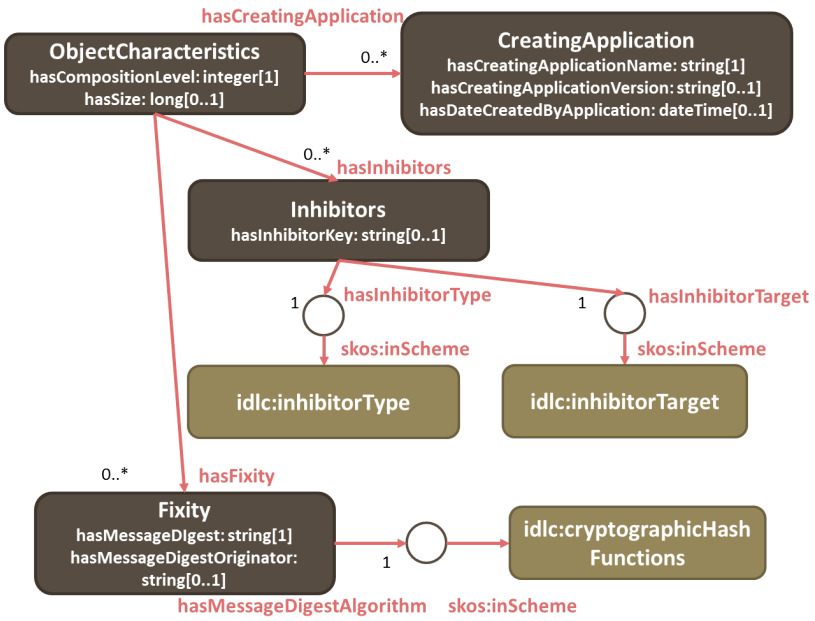

Figure 2.16: Overview of the *CreatingApplication*, *Fixity*, and *Inhibitor* Classes.

2.4.4 Event

An event aggregates all the information about an action that involves one or more objects. This metadata is stored separately from the object metadata. Actions that modify objects should always be recorded as events.

The *Event* class, shown in Figure 2.18, is described at least by an *eventType* and an *eventDateTime*. The *eventType* values are taken from the preservation vocabulary idlc:eventType. This information can be extended using the *eventDetail* property, which gives a more detailed description of the event, and the *eventOutcomeInformation*, which describes the outcome of the event, in terms of success, failure, partial success, etc.

2.4.5 Agent

This class aggregates information about attributes or characteristics of agents. Agents can be persons, organisations or software. This class provides the necessary tools to identify unambiguously an agent. The minimum property needed to describe the *Agent* class is *hasAgentType*.

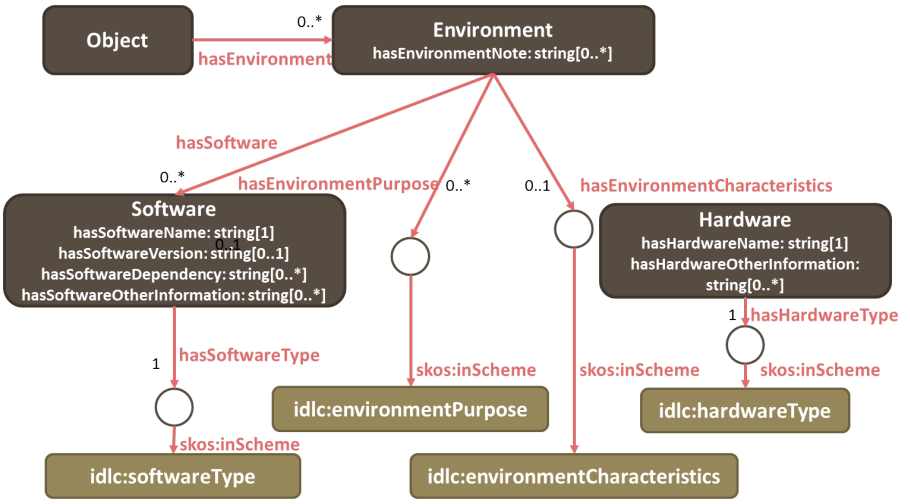

Figure 2.17: Overview of the *Environment* Classes.

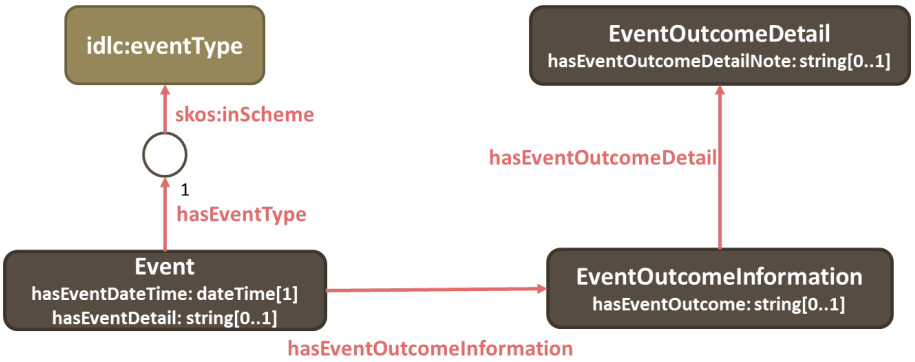

Figure 2.18: Schematic description of the *Event* Class.

The values of this property are taken from the preservation vocabulary idlc:agentType. Optionally, an agent can also be described using the *hasAgentName* and *hasAgentNote*. The *Agent* class is a subclass of *foaf:Agent* and *dcterms:Agent*. In Figure 2.19, a schematic overview of the *Agent* class is depicted.

2.4.6 Rights

PREMIS also foresees to describe rights. The minimum core rights information that a preservation repository must know, however, is what rights or permissions a repository has to carry out actions related to objects within the repository. The rights or permissions of certain agents

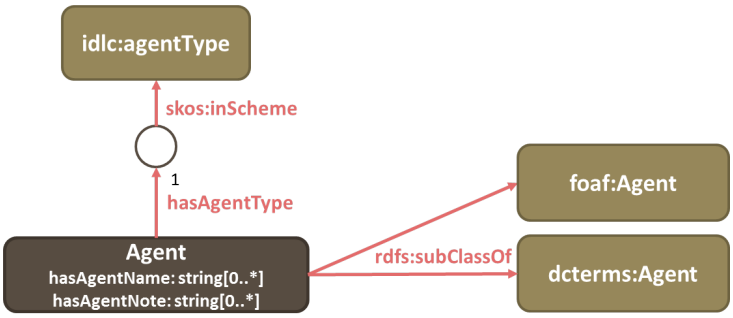

Figure 2.19: Schematic Description of the *Agent* Class.

may generally be granted by copyright law, by statute, or by a license agreement with the rightsholder. For this, the *RightsStatement* class knows three subclasses, denoting the rights basis for the rights statement: *CopyrightInformation*, *LicenseInformation*, and *StatuteInformation*, as depicted in Figure 2.20. In some situations the basis for the rights is for other reasons, for instance institutional policy. If the basis for the rights is different, one can introduce its own subclass to *RightsStatement*. The *RightsStatement* class on itself is a subclass of *dcterms:RightsStatement*.

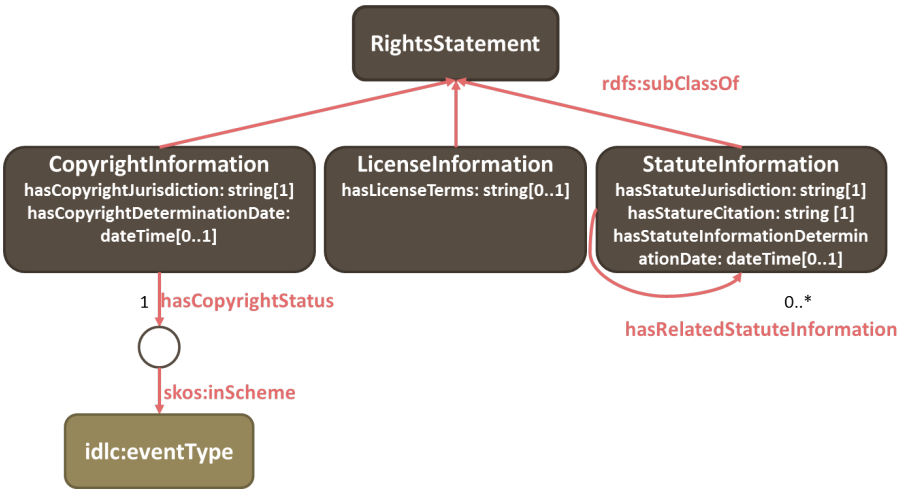

Figure 2.20: Subclasses of the *RightsStatement* Class.

Documentation of the rights can be attached to a *RightsStatement* instance using the *RightsDocumentation* class. A *RightsStatement* instance can be further characterised by the dates it is applicable using the *ApplicableDates* class. And, it allows describing the granted rights using the *RightsGranted* class. This *RightsGranted* class is able to denote the

period the actions that are granted using the *hasAct* property. The values for this property are taken from the preservation vocabulary idlc:actionsGranted. In Figure 2.21 these details on the *RightsStatement* class are shown.

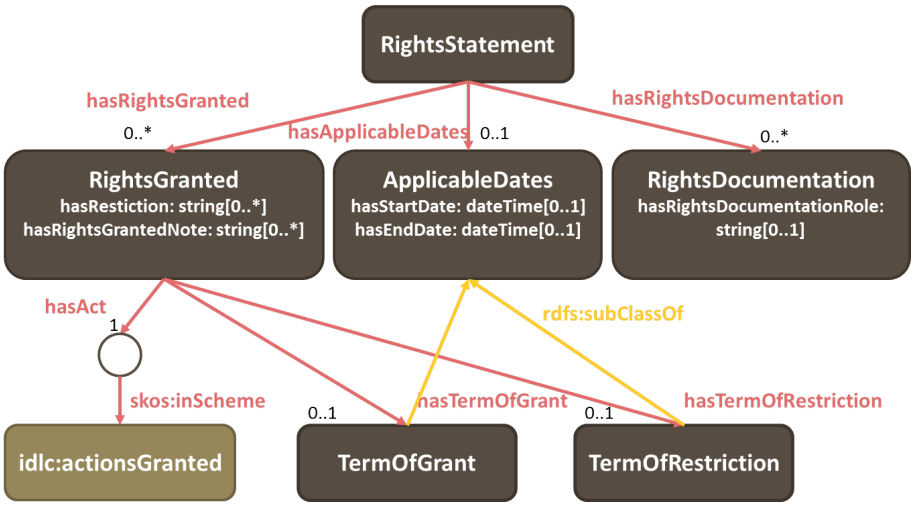

Figure 2.21: Details of the *RightsStatement* Class.

2.5 Validation

The validation of PREMIS OWL follows the a workflow which is also used at W3C. PREMIS OWL is designed by a restricted group of domain experts from the LOC, BNF, Familysearch and iMinds. During weekly meetings PREMIS OWL has been designed. After this design process, the ontology and its design decisions were documented and published on a public wiki, for which PBWorks[4] is used. This wiki allowed to interact with domain experts and to take their comments into consideration to refine the ontology. This way, two PREMIS OWL revisions have been created so far. Meanwhile the LOC has assembled a working group around PREMIS OWL to keep the ontology aligned with the PREMIS Data Dictionary, which is still evolving. This working group consists of members of the PREMIS Editorial Committee, members of LOC, BNF and is open for other interested domain experts to maintain the ontology.

[4] `http://premisontologypublic.pbworks.com/w/page/45987067/FrontPage`

2.6 Conclusions

In this chapter, I have presented PREMIS OWL, an ontology based on the PREMIS Data Dictionary for Preservation Metadata version 3.0, a digital preservation standard based on the OAIS reference model. The ontology was first designed within the project Archipel, initiating the long-term preservation archivation in Flanders. Later on, this ontology was picked up by the Library of Congress. After several years of cooperation with the Library of Congress, the Bibliothèque nationale de France (BNF), and Family Search in refining the ontology, it is now supported by the Library of Congress. At the moment, the ontology is published, and maintained by the Library of Congress, as their official semantic binding of the PREMIS.

The ontology tries to stick as closely as possible to the PREMIS Data Dictionary semantic unit definitions. The ontology is ideal for creating, validating and storing the preservation metadata of a particular digital asset. Before, the PREMIS Data Dictionary was only implemented as an XML schema. This semantic binding of PREMIS does not replace the XML schema, it is complementary to it.

The main difference of the ontology to the data dictionary is the introduction of 24 preservation vocabularies, formalised as SKOS vocabularies, also published by the Library of Congress. This feature greatly enhances the interoperability of PREMIS OWL instances coming from different institutions. Because every archiving institution has its own preservation policy and preservation services, the ontology needs to be extensible to reflect these institutional differences. In the XML version of the data dictionary, every institution could use their own controlled vocabularies. This feature prohibited interoperability between different archiving institutions. In the ontology, the use of proprietary vocabularies is still possible, but they must be linked to the preservation vocabularies to guarantee interoperability.

This OWL ontology allows one to provide a Linked Data-friendly, PREMIS-endorsed serialization of the PREMIS Data Dictionary version 2.2. This can be leveraged to have a Linked Data-friendly data management function for a preservation repository, allowing for SPARQL querying. It integrates PREMIS information with other Linked Data compliant datasets, especially format registries, which are now referenced from the PREMIS ontology (for instance, the Unified Digital Format Registry and PRONOM). Thus information can be more easily interconnected, especially between different repository databases.

The author's work on the design and formalisation of this ontology, and in general around the modelling and representation of provenance information led to the following publications:

- Erik Mannens, Ruben Verborgh, Seth Van Hooland, Laurence Hauttekeete, Tom Evens, Sam Coppens, and Rik Van de Walle. „On the Origin of Metadata". In: *Information* 3.4 (2012), pp. 790–808
- Sam Coppens, Jan Haspeslagh, Patrick Hochstenbach, Erik Mannens, Rik Van de Walle, and Inge Van Nieuwerburgh. „Metadatastandaarden, Dublin Core en het gelaagd metadatamodel". eng. In: ed. by Stoffel Debuysere, Dries Moreels, Rik Van de Walle, Inge Van Nieuwerburgh, and Jeroen Walterus. Bewaring en ontsluiting van multimediale data in Vlaanderen : perspectieven op audiovisueel erfgoed in het digitale tijdperk. Lannoo Campus, 2010, pp. 46–62. ISBN: 9789020989441
- Erik Mannens, Sam Coppens, and Rik Van de Walle. „Semantic BRICKS for performing arts archives and disseminiation". In: *IASA JOURNAL* 35 (2010), pp. 40–49
- Sam Coppens, Erik Mannens, and Rik Van de Walle. „Disseminating heritage records as linked open data". In: *International Journal of Virtual Reality* 8.3 (2009), pp. 39–44
- Luc Moreau, Paolo Missier, Khalid Belhajjame, Reza B'Far, James Cheney, Sam Coppens, Stephen Cresswell, Yolanda Gil, Paul Groth, Graham Klyne, Timothy Lebo, Jim McCusker, Simon Miles, James Myers, Satya Sahoo, and Curt Tilmes. *PROV-DM: The PROV Data Model*. Tech. rep. URL: `http://www.w3.org/TR/prov-dm/`
- Sam Coppens and Tom De Nies. *PROV-Dictionary: Modeling Provenance for Dictionary Data Structures*. Tech. rep. 2013. URL: `http://www.w3.org/TR/prov-dictionary/`
- Yolanda Gil, James Cheney, Paul Groth, Olaf Hartig, Simon Miles, Luc Moreau, Paulo Pinheiro Da Silva, Sam Coppens, Daniel Garijo, JM Gomez, et al. „Provenance XG Final Report". In: (Dec. 2010). URL: `http://www.w3.org/2005/Incubator/prov/XGR-prov/`
- Satya Sahoo, Paul Groth, Olaf Hartig, Simon Miles, Sam Coppens, James Myers, Yolanda Gil, Luc Moreau, Jun Zhao, Michael Panzer, et al. *Provenance vocabulary mappings*. Tech. rep. 2010. URL: `http://www.w3.org/2005/Incubator/prov/wiki/Provenance_Vocabulary_Mappings`

- Sam Coppens, Sebastien Peyrard, Rebecca Guenther, Keving Ford, and Tom Creighton. *PREMIS OWL ontology 2.2*. Tech. rep. 2013. URL: http://www.loc.gov/standards/premis/ontology-announcement.html
- Sam Coppens, Sebastien Peyrard, Rebecca Guenther, Keving Ford, and Tom Creighton. *PREMIS OWL ontology 2.1*. Tech. rep. 2011. URL: http://www.loc.gov/standards/premis/owlOntology-announcement.html
- Tom De Nies, Sara Magliacane, Ruben Verborgh, Sam Coppens, Paul Groth, Erik Mannens, and Rik Van de Walle. „Git2PROV: Exposing Version Control System Content as W3C PROV". in: *Poster and Demo Proceedings of the 12th International Semantic Web Conference* (2013)
- Sam Coppens, Erik Mannens, Raf Vandesande, and Rik Van de Walle. „Digital long-term preservation, provenance and linked open data". In: *2011 European Library Automation Group conference (ELAG 2011): It's the context, stupid!* 2011
- Sam Coppens, Erik Mannens, and Rik Van de Walle. „What about the provenance of media archives: a tale of time-contextualisation". In: (2011)
- Sam Coppens, Erik Mannens, and Rik Van de Walle. „PREMIS OWL binding to workflow engine for digital long-term preservation". In: (2010)
- Sam Coppens, Erik Mannens, Tom Evens, Laurence Hauttekeete, and Rik Van de Walle. „Digital long-term preservation using a layered semantic metadata schema of PREMIS 2.0". In: *Cultural heritage on line, Florence, 15th-16th December* (2009)

Chapter 3

Archipel - A Distributed, Digital Long-term Preservation Infrastructure

Nowadays, the need for digital preservation is growing, since a lot of the digital information produced merely a decade ago is in danger of getting lost as technologies are changing. This also threatens a lot of information from heritage institutions. The research project Archipel investigates how to overcome the threat of information loss for cultural heritage institutions in Belgium, due to the lack of a proper archiving strategy. In this project, we developed a platform that harvests the information from the institutions, preserves the information for the long-term and disseminates the information as Linked Open Data. Our platform has all the necessary processes to guarantee the long-term preservation of the incoming data. For the platform, we used a service-oriented architecture, where all the preservation services are implemented as services on an enterprise service bus. The built-in workflow engine will orchestrate all the services to fulfill the two objectives of the platform, i.e., the LOD dissemination and the long-term preservation. This service oriented architecture allows to build a distributed long-term preservation platform supporting cloud storage in its back-end.

3.1 Introduction

Many organisations and private persons still possess a lot of material which is stored on analogue carriers. This material is mostly part of important cultural heritage anywhere. At this moment, the analogue

carriers are degrading and continuously losing quality, making the data inaccessible. While we are still able to see wall paintings from millennia ago, many documents from merely a decade or two decades ago have become inaccessible, e.g., *WordPerfect* files. Some refer to this situation as the *Digital Dark Age*[82]. Digital long-term preservation forms the solution to this issue. A digital long-term archive has the necessary processes in place to withstand many long-term preservation risks, e.g., bitrot, file formats becoming obsolete, etc. These preservation processes make sure the content remains intact and accessible over time.

The project *Archipel*[1] initiates the dissemination and digital long-term preservation of the cultural heritage in Flanders, Belgium. For this, we developed a platform that harvests data coming from various institutions (libraries, archival institutions, the art sector, i.e., museums, and the broadcasters), preserves the data for the long-term and disseminates the data as LOD. Our developed platform has thus two main objectives:

- Dissemination of content and its provenance as LOD
- Digital long-term preservation of content

To guarantee the long-term preservation of the harvested content, our platform has the necessary processes in place to keep the information intact and interpretable, in line with the Open Archival Information System (OAIS) reference model [83] for the long-term preservation of information. These processes rely heavily on the provenance information of the harvested data, but at the same time produce also a lot of provenance information. This provenance information is modeled using a semantic implementation of the PREMIS 3.0 data dictionary [7], i.e., PREMIS OWL[2]. This ontology is supplemented with a semantic Dublin Core[3] layer in our layered semantic metadata model. This enables the archive to deal with the very diverse metadata it has to accept for preservation. This Dublin Core layer is not only used as a common ground for managing the content, but also for the Linked Open Data [14] (LOD) publication of the records.

In this chapter, we present our digital long-term preservation platform. First, we describe some related work on the topic in Section 3.2. Then, in Section 3.3 we introduce the OAIS reference model for long-term preservation. This reference model forms the basis for our deep archive and describes the necessary processes and functions of our archive to

[1] `http://www.archipelproject.be`
[2] `http://www.loc.gov/premis/rdf/v1#`
[3] `http://dublincore.org/`

deal with the preservation risks, described in Section 2.2. Next, we discuss the requirements of the archive in Section 3.4. In Section 3.5, we discuss the format and the content of our *information packages*, our archive has to deal with. Section 3.6 will describe the distributed architecture of the archive and its processes. We end with a conclusion in Section 3.7.

3.2 Related Work

Interest in digital preservation can be seen by the multitude of projects in this area. Planets (Preservation and Long-term Access through Networked Services)[4] was especially aimed at defining guidelines for preservation planning. However, it did not tackle the integration of different existing metadata formats, or the dissemination of the metadata as LOD. Likewise, the Prestospace (Preservation towards storage and access) project's objective was to provide technical solutions and integrated systems for a complete digital preservation of all kinds of audio-visual collections [5]. The project was especially focused on the underlying technologies, e.g., automated generation of metadata or detection of errors in content [84], but without using a standardised, semantic preservation model to support the archiving, nor do they tackle the problem of publishing the generated provenance information to the Web. The CRiB system [85] delivers a set of services that client applications will be able to invoke in order to perform complex format migrations, evaluate the outcome of those migrations according to multiple criteria (e.g., data loss and performance), and obtain detailed migration reports for documenting the preservation intervention. It is aimed at supporting the migrations of the archived content, but it does not tackle problems like the LOD publication the archived content and their provenance information. PANIC [86], An Integrated Approach to the Preservation of Composite Digital Objects using Semantic Web Services, is an integrated, flexible system, which leverages existing tools and services and assists organizations to dynamically discover the optimum preservation strategy for compound objects. The system has a service-oriented architecture and the Web services are semantically described, allowing to discover the most appropriate preservation strategy for the compound object or its aggregated objects. But, as the CRiB system, it does not focus on the publication of the content nor on the publication of the provenance information.

[4] http://www.planets-project.eu/

[5] http://prestospace.org/project/index.nl.html

The CASPAR project (Cultural Artistic and Scientific knowledge for Preservation, Access, and Retrieval) presented technologies for digital preservation [6]. The OAIS Reference Model was chosen as the base platform, and the project was focused on implementing the different steps in the preservation workflow. They focus more on preservation services than on describing the preservation information. BOM Vlaanderen [7], a national research project, aimed at preservation and disclosure of audio-visual content in Flanders. Additionally, it looked at ways to unify different metadata standards currently used for describing audio-visual content. Current trends are on integrating different media archives. PrestoPRIME has investigated and developed practical solutions for the long-term preservation of digital media objects, programmes and collections, and finds ways to increase access by integrating the media archives with European on-line digital libraries in a digital preservation framework[8].

The previous discussed related work were focusing on the digital long-term preservation, not on the more general problem of enabling their provenance information on the Web. For the work done in this area, the work of the W3C Provenance Incubator Group[9] is the major reference. This incubator group produced working definitions for provenance information, provided a state-of-the-art understanding and developed a roadmap for development and possible standardisation of provenance on the Web. This work included defining key dimensions for provenance, collecting use cases, designing three flagship scenarios from the use cases, creating mappings between existing provenance vocabularies, looking how provenance could fit in the Web architecture and providing a state-of-the-art report on the current provenance activities. Their work is summarised in a final report [87]. The first flagship scenario describes a news aggregator site that assembles news items from a variety of data sources, e.g., news sites, blogs and tweets. The provenance records of these data providers can help with verification, credit and licensing. This flagship scenario could be covered by publishing the provenance information using our framework. What still forms a problem is the lack of a standardised metadata model for publishing provenance on the Web. In our framework, we publish the provenance information

[6] http://www.casparpreserves.eu/

[7] https://projects.ibbt.be/bom-vl

[8] http://www.prestoprime.org/

[9] http://www.w3.org/2005/Incubator/prov/wiki/W3C_Provenance_Incubator_Group_Wiki

as Linked Open Data using PREMIS OWL. This information is only interoperable in the long-term preservation context, where PREMIS is well known, not in a Web context. This standardised provenance model for the Web is still a major research area. The work of the W3C Provenance Incubator Group was a first step into that direction. The W3C Provenance Working Group finalised a specification for provenance on the Web. In both groups I was a major contributor, representing the archiving domain.

Another interesting work done in the area of publishing provenance for linked data is the paper of Olaf Hartig and Jun Zhao published at IPAW [88]. In that paper they describe the Provenance Vocabulary[10] used for describing the provenance information as Linked Open Data. Next to this, they also offer ways of publishing this provenance information for Linked Data. They discuss how provenance can be added to Linked Data objects, how provenance can be included into RDF dumps and how the provenance information can be queried using SPARQL endpoints. This work enables provenance for Linked Data, but it does not offer solutions for automatic discovery of the provenance information or ways for publishing provenance on the Web beyond using semantic web technologies. Future work could involve publishing the provenance information using this vocabulary, which is more suited for publication on the Web than PREMIS OWL, which is intended to be a data model for digital long-term archives. The mapping table, relating various provenance vocabularies, produced by the W3C Incubator Group[11] will be the reference for this work.

3.3 OAIS

The Open Archival Information System (OAIS) is an archive that preserves the information and makes it available to a designated community. This designated community is the identified group of potential consumers of the archived information. They should always be able to understand the information. These consumers may consist of multiple communities and may change over time. Next to the consumer, or designated community, we also have the producers, from which the data comes that has to be preserved. These actors are depicted in Figure 3.1. The OAIS reference model, depicted in Figure 3.2, is a basis model

[10] http://purl.org/net/provenance/

[11] http://www.w3.org/2005/Incubator/prov/wiki/Provenance_Vocabulary_Mappings

Figure 3.1: Actors of an Archive

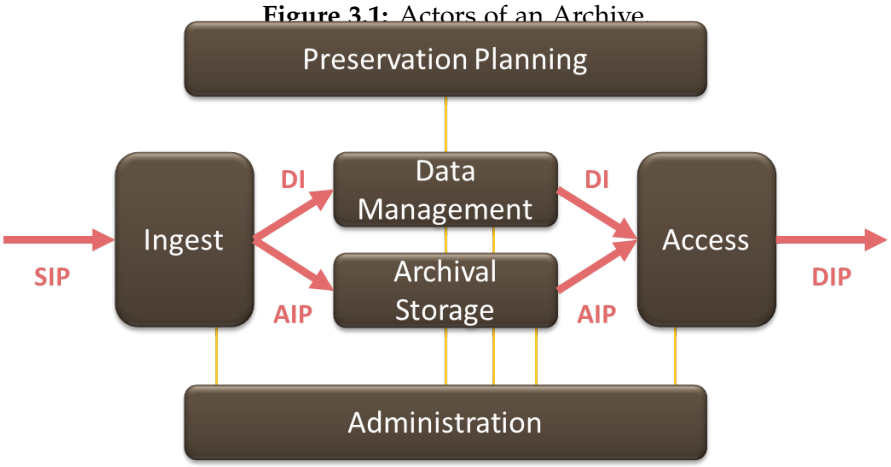

Figure 3.2: OAIS Reference Model.

for long-term archives. It assures platform-independent access to all archived information. The reference model (ISO 14721:2003) includes the following responsibilities that an OAIS archive must abide by:

- Negotiate for and accept appropriate information from information Producers.
- Obtain sufficient control of the information provided to ensure Long-Term Preservation.
- Define your designated community.
- Ensure that the information to be preserved is independently understandable to the designated community. Thus, the community should be able to understand the information without needing the assistance of the experts who produced the information.
- Follow documented policies and procedures which ensure that the information is preserved against all reasonable contingencies, and which enable the information to be disseminated as authenticated copies of the original, or as traceable to the original.
- Make the preserved information available to the designated community.

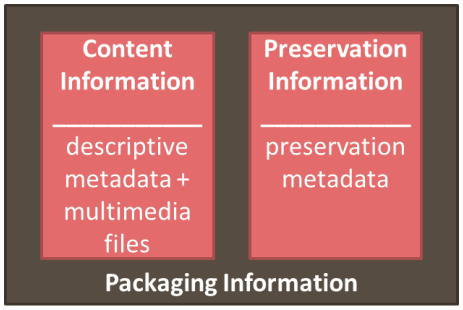

Figure 3.3: Archival Information Package of OAIS.

In general, it describes the digital archive using three types of packages. Each package aggregates the needed metadata, according to its function, and the digital files accompanying the metadata, shown in Figure 3.3. As depicted in Figure 3.2, these packages are:

- **Submission Information Package** (SIP): This is the package the archives accept for ingest. It consists of descriptive metadata, digital multimedia files, referenced by the metadata and some additional metadata, e.g., structural metadata or rights metadata.
- **Archival Information Package** (AIP): This is the package the archives use for preservation. It is in fact the SIP supplemented with preservation information.
- **Dissemination Information Package** (DIP): This is the package the archives offer for dissemination, e.g., if the rights do not permit to publish the archived digital files accompanying the metadata, this package will only contain metadata.

Using these packages in the operational mode of the archive, makes sure the content can always be pulled out of the content management system and put into a new content management system. For the management of these packages, the reference model uses the descriptive metadata of the object, denoted as DI, i.e., descriptive information, in our Figure 3.3. Basically, this should be the default architecture of every digital repository. What makes a digital repository a long-term preservation platform are the preservation plans. These preservation plans implement the needed processes/workflows to keep the information intact and accessible. They are based on the policies of the archive.

3.4 Requirements of the Archive

First, we need to define the form of our information packages, i.e., SIP, AIP and DIP. These packages consist of descriptive information on the object to be preserved, the referenced multimedia files, preservation information, and packaging information. For this, we need to know who the producers of the data are and the designated community. The producers will define the descriptive metadata model for the SIPs and the AIPs. The designated community will define the descriptive metadata model of our DIPs.

Next, we need to know the policies of the archive to be designed. These policies will define the preservation planning. The preservation planning will eventually define the services and the workflows, based on these services, to guarantee the information remains intact and interpretable by the designated community. Thus, the actors of the long-term preservation platform will define the data model, and the preservation plan will define the architecture of the archive. Both are discussed in detail in the next sections.

To end, we give a short overview of the requirements of our archive. The producers are in fact institutions from the heritage sector. This sector includes broadcasters, libraries, archives, and museums. The designated community that will be targeted consist of different dissemination modules:

- Performing Arts: *Toneelstof* is a dissemination module that will put up a website around the history of performing arts.
- Media Education: Another dissemination module will focus on media education. For this, *Ambrosio's Tafel* has developed a platform to build up the know-how and expertise on media. The platform will tap into the long-term preservation archive to dig up its content.
- Artistic Reuse: *Constant VZW* has developed a video wiki, based on the archive's content, to promote artistic reuse of the archived content. The wiki is called Active Archives Video Wiki.
- Resource Sharing: *Klascement*, a Belgian educational platform, will take the archive's content to promote resource sharing on their educational platform between teachers.
- Research: *The Boekentoren*, university library of the Ghent university, has developed a platform to foster research in the archived content. For this, they developed an image bank, public for the researchers.

Looking at the preservation policies or the preservation plan, the following requirements can be listed:

- Content needs to remain intact: for this, we will need integrity checks on the files that are preserved.
- Content needs to remain interpretable by the designated community: to this end, we need to define the metadata model of the descriptive metadata disseminated by our archive, and the file formats that are accepted by our archive and the file formats in which the referenced multimedia files will be disseminated so they remain interpretable.
- Dissemination as Linked Open Data
- Distributed Architecture, supporting cloud storage.

What needs to be defined before moving on to the implementation of our platform is the form and content of the information packages, and the preservation processes that will fulfil the requirements of the platform.

3.5 Information Packages

First, the information packages need to be modeled. This means specifying the packaging format and the contents of these packages.

3.5.1 Packaging Format

As a packaging format for our **SIPs** and **DIPs**, *BagIt* is used. These packages keep the metadata and the referenced files together in a package. They provide an explicit link between the metadata and their multimedia files. *BagIt* allows storing the metadata together with the files referenced by the metadata, and putting these data in a *ZIP* archive for transportation to and from our preservation platform. This package also allows storing some fixity information in it, e.g., MD5 checksums for validation. In our case, the **SIP** will consist of references to the multimedia files, together with the original metadata offered by the cultural institution. Thus, our SIP has the following payload files in its package:

- The original metadata record, e.g., MARC or CDWA.
- The files accompanying the metadata.

The resulting *BagIt ZIP* package contains the following tag files and a directory containing the payload files:

- bagit.txt: this file contains general information regarding the *BagIt* version used and the character encoding.
- bagit-info.txt: This file contains general information regarding the *BagIt* size, the date the *BagIt* package was made, the octetstream sum, etc.
- manifest-[algorithm].txt: this file lists the payload files and corresponding checksums generated using a particular algorithm, denoted in the file name of this tag file.
- tagmanifest-[algorithm].txt: this file lists the *Bagit* files, not the payload files, and their corresponding checksums.
- data: this is the folder containing the payload files.

The **DIP** will consist of references to the multimedia files, together with the original metadata offered by the cultural institution, the mapped metadata, and the transcoded multimedia files.

Thus, our DIP has the following payload files in its package:

- The original metadata record, e.g., MARC or CDWA.
- The mapped metadata record.
- The multimedia files accompanying the metadata.
- The transcoded multimedia files.

3.5.2 Package Content

The packages' content refers to the metadata models used for the packages, i.e., the metadata models for the descriptive metadata of the SIPs, AIPs, and DIPs, and the metadata model for the preservation metadata. The packages' content also refers to the file formats of the multimedia files of our packages. The file formats the archive accepts and the ones they use for disseminating the preserved content to their designated community.

A. Metadata Models

Descriptive metadata schemes describe the content of the data: subject, author, date of creation, file format, etc. This metadata makes it possible to manage and search the digital archive as a whole, cfr. Section 3.3. When archiving data coming from different sectors like the broadcast sector, the libraries, the cultural sector, and the archival sector, a problem arises concerning descriptive metadata. Many of the institutions already have descriptive metadata. Are these descriptive metadata stored as metadata or as data? When archiving these descriptions as

metadata, the archive has to provide a metadata schema. The choice of this schema is a non-trivial task, as the metadata schemes used for the descriptions are very domain-specific. To store the descriptive metadata lossless, the descriptive metadata schema should be some kind of smallest common multiple of all the descriptive metadata schemes offered by the different institutions. This would be a huge metadata schema, impossible to maintain. That is why the descriptive metadata is archived along with the data in their original metadata format, e.g., MARC , so there is no information loss. On top of this domain-specific metadata, the archive offers a broadly accepted descriptive metadata schema. This gives the archive the necessary tools to search the whole archive. When finding the data of interest, the original metadata that is stored as data can still be presented to the users.

For the demonstration of our platform, we currently accept the following descriptive metadata formats, which are the metadata models we accepts in the offered **SIPs**:

- *Dublin Core*: mostly used metadata model, supported by many repositories.
- *MARC21*: this metadata format is mainly used by the libraries.
- *EAD* and *EAC*: these metadata formats are designed to be used in the archiving sector.
- *P/META*: this metadata format is geared to support the broadcasters.
- *CDWALite*: this descriptive metadata format is a domain ontology for the arts sector (museums, etc.).

This design choice also facilitates the preservation of the metadata. As described in Section 2.4, PREMIS is only capable of describing the preservation information of digital multimedia objects. The events in PREMIS only take place on objects, not on intellectual entities, which encapsulate the descriptive metadata. By storing the descriptive metadata apart like any other data stream, such a description of the intellectual entity also becomes a separate object, for which PREMIS can describe the preservation information.

DC RDF [89] was chosen as format for the descriptive metadata to use internally (**AIP**) and for its dissemination (**DIP**), as it is a broadly accepted descriptive schema. The power of this schema is its simplicity and generality. It only consists of fifteen fields among which creator, subject, coverage, description, and date. It can answer to the basic questions: Who, What, Where, and When. All the fields in DC are

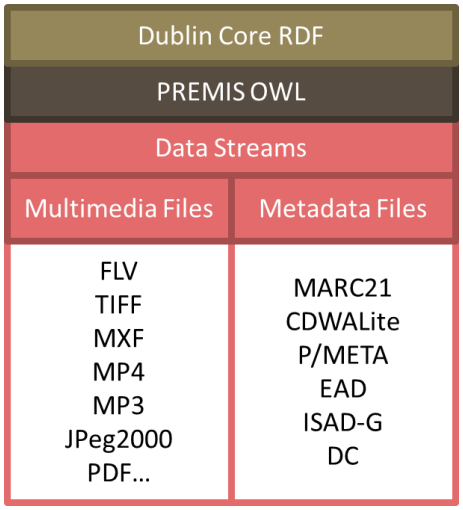

Figure 3.4: Layered data model for the long-term archive.

optional and repeatable. This makes it possible to map relatively easily almost all the descriptive metadata schemes to DC RDF as many institutions already support DC. This leads to a layered, semantic metadata model, which the archive uses for management, dissemination and preservation purposes, as depicted in Figure 3.4. Similar work is described in [90], where a core ontology was developed to harmonise the knowledge perspectives of the libraries and the cultural heritage institutions, resulting in a core ontology.

B. File Formats

Next, we need to define the file formats the archive will handle. Our archive will handle any file format delivered to it (**SIP**), without any restriction. The file formats we are going to use internally (**AIP**) and for the dissemination (**DIP**) can be different. The file formats for the dissemination will be determined by its support in modern browsers. The choice for the file formats internally is based on two factors: it must be lossless or near lossless and we have to keep into account the available storage space.

For demonstration purposes, the following file formats are agreed:

- One *dissemination* migration service for transforming text documents to PDF documents.
- One *master copy* migration service for transforming images to JPEG2000 images.
- One *master copy* migration service for transforming sound to AAC files.
- One *dissemination* migration service for migrating video to MP4 files.

3.6 Architecture

In this Section, our architecture of the digital long-term preservation archive is described. In this networked world, various resources are linked to each other. We do not want to build yet another central e-depot, but a distributed network of storage components. For this reason, the platform will have a service oriented architecture[12] (SOA). This SOA will make use of a central service hub, which will offer the needed services for the platform. The objectives of our platform are twofold:

- Disseminate the archived content and their provenance as LOD.
- Enable long-term preservation.

Our architecture is depicted in Figure 3.5. The vertical arrow indicates the dissemination path, the horizontal arrow stipulates the preservation path. The basic components of our architecture are:

- *Repositories*: these are the repositories of the diverse institutions, which have their content published on-line, using the OAI-PMH protocol [91].
- *Shared Repositories*: for those institutions, which do not have published their content on-line, our Archipel project foresees several shared repositories, using *Omeka*[13] or *MediaMosa*[14], which will publish their content on-line using the OAI-PMH protocol.
- *Integration Server*: this server provides an integration layer for orchestrating all the needed processes, which are all implemented as web services, e.g., transcoding services.

[12] http://opengroup.org/projects/soa/
[13] http://omeka.org/
[14] http://www.mediamosa.org/

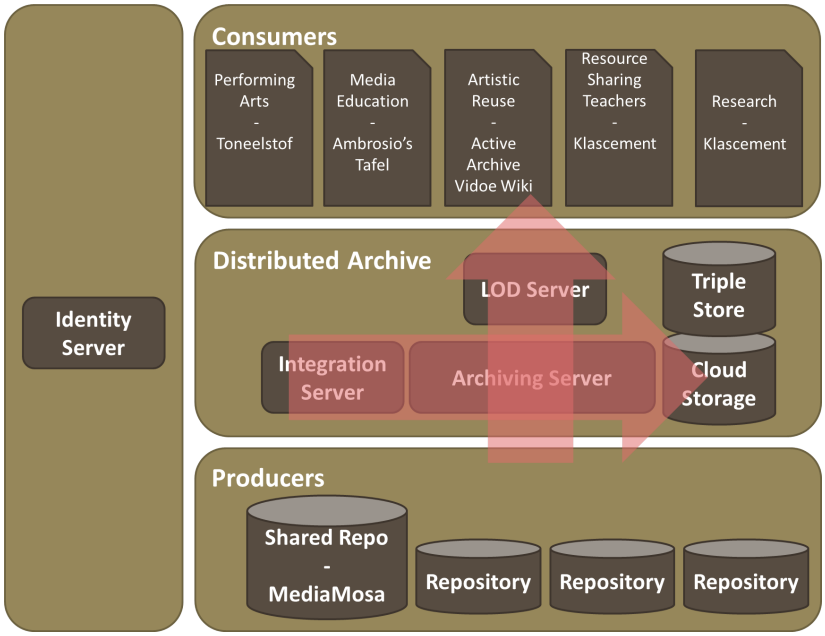

Figure 3.5: Architecture of the long-term preservation platform.

- *Archiving Server*: This server actually has a direct connection to the triple store for the descriptive and preservation metadata and the distributed storage for the referenced multimedia files. The services are actually hosted here, close to the data they act on. The services are orchestrated by the integration server.
- *LOD server*: this server is used for the dissemination of the content and the provenance information, with a triple store as a storage back-end.
- *Identity Service*: with this distributed architecture an identity server is needed for authentication across the different systems.

3.6.1 Integration Server

For building our distributed, digital long-term preservation platform, we need an integration server to orchestrate the different processes, based on SOA technology. This service-oriented approach consists of flexible granular functional components that expose service behaviours to other applications via loosely coupled standards-based interfaces. Our architecture needs to handle many different preservation and dissemination processes, which change over time as technologies change. As such, the situation demands for a service-oriented approach, which allows to expand and modify the current setup of our processes.

An Enterprise Service Bus (ESB, [92]) provides the open, standards-based connectivity infrastructure for the service oriented architecture and allows these services to exchange data with one another as they participate in our processes. Orchestration between services is handled by a workflow engine. This engine is integrated in the service bus architecture and supports the execution of the preservation processes. For the communication, the 'Simple Object Access Protocol' (SOAP) [93] is used, a protocol specification for exchanging structured information between services. This integration server is built using the *Porthus*[15] *.NET* Integration server, as *Porthus* was a project partner. A better choice for the integration would have been a BPEL engine, which is more suited for long-running and complex business processes, but the use of the *Porthus* integration server was a requirement of the project.

The whole preservation / dissemination cycle, starts with a **harvesting process**, which will harvest the metadata, and the referenced files. The harvested metadata are described using several descriptive metadata formats, e.g., MARC21, DC, or CDWALite. For management and dissemination purposes this metadata needs to be mapped to DC RDF. For this, we rely on a **mapping service**, which will map the incoming metadata to DC descriptions.

Next, the original metadata record, the mapped DC RDF record and the referenced files get packed into a SIP. For this SIP, the *BagIt* package format is used. This SIP package is then ingested into the platform, using the **SIP ingest service**. Next to the harvesting (pull), the producers can also directly push SIP packages to the archiving server. This data enters immediately the workflow using the *SIP ingest service*.

At ingest the SIP packages need to be extended with provenance information to form AIPs. This is done by the **characterisation service**. It identifies the file formats of the files packaged in the SIP and models these files as PREMIS *Object* instances. The SIPs get extended with their PREMIS *Object* instances to become AIPs. Every action performed on such a PREMIS *Object*, will be modelled as a PREMIS *Event* and will get related to that *Object*.

[15] `http://www.porthus.be`

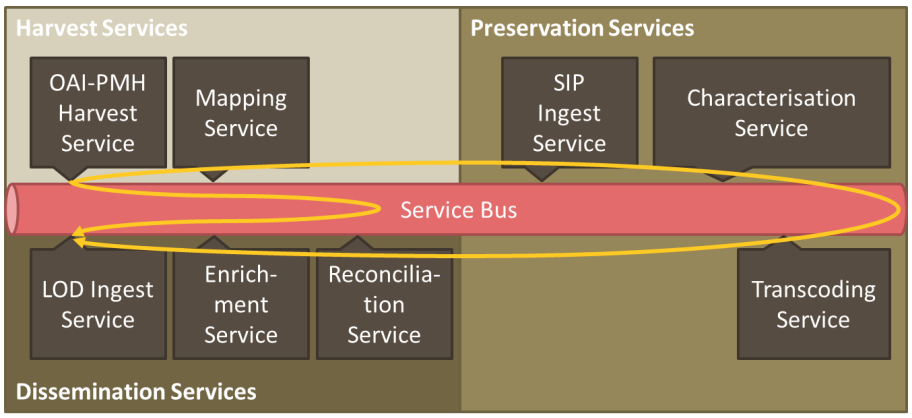

Figure 3.6: Schematic Overview of the Service Bus and its Connected Services.

The next thing within the workflow is the migration of the stored, related multimedia files. These files get migrated to a file format, defined by the archives preservation plans. Such a preservation plan can stipulate, e.g., that all image files must be migrated to the JPEG2000 file format to keep the image information accessible. For this, we need **transcoding services**, which can then transform various incoming image file formats to the JPEG2000 file format. This migration will extend the AIP package with the extra migrated data stream. This data stream is then passed to the characterisation service to get a PREMIS *Object* description of the generated data stream. The preservation information is also extended with a description of the migration service as a PREMIS *Event* relating the source object to the migrated object.

During the last phase, the descriptive DC RDF metadata will get reconciled and cleaned by the *reconciliation service* and enriched by the **enrichment service**. This service will interlink the data with internal and external data sources.

If the harvested content does not need to be preserved, but only needs to be published as LOD, it is directly routed from the mapping service to a *reconciliation service*, and then to our **enrichment service**. These enriched DC descriptions then get ingested into our triple store. This is done by the **LOD ingest service**. Our LOD server automatically publishes these enriched DC records as LOD. This whole preservation / dissemination path is described in Figure 3.6. The individual services will be discussed in more detail during the next sections.

3.6.2 Harvest Services

The harvest services play a role in harvesting the data from the institutions' repository or from the shared repositories, used by smaller institutions, who have not published their content on-line yet. They will pull in the content for both dissemination and preservation.

A. OAI-PMH Harvest Service

This service will harvest the content from the cultural institutions. The protocol used for harvesting the data is OAI-PMH, a wide-spread protocol for exchanging metadata, particularly popular in the domain of digital libraries. The OAI-PMH protocol disseminates the metadata about the items of a repository. These items describe digital or non-digital resources, and are identified by a URI [11]. Each item can have multiple metadata records, each of which is described by a certain metadata schema. These schemes are chosen by the data provider to suit their domain-specific demands. To guarantee a basic level of interoperability one of those metadata schemes must be unqualified DC. Figure 3.7 shows the basic concepts of the protocol.

The OAI-PMH protocol is based on HTTP [12]. It supports six request types (verbs). The request arguments are issued as GET or POST parameters. The responses are encoded in XML syntax. The *Identify* request retrieves administrative metadata about the repository, e.g., the name or owner of the repository; *GetRecord* retrieves metadata about an item in a certain metadataformat; *ListRecords* harvests all metadata records in a certain metadata format for all items in the repository; *ListIdentifiers* lists all the identifiers of the available items; *ListMetadataFormats* returns the available metadata formats used in the repository; and finally the sixth verb, *ListSets* gives the available sets in the OAI-PMH repository.

B. Mapping Service

The harvesting service has to be able to deal with multiple metadata formats. For this, we need a common ground for the metadata for search and retrieval purposes. This common ground is DC RDF, the top layer of our layered metadata model. The mapping will be done using a central mapping service, which is able to accept certain metadata records and convert them to DC RDF using XSLT [94] transformations.

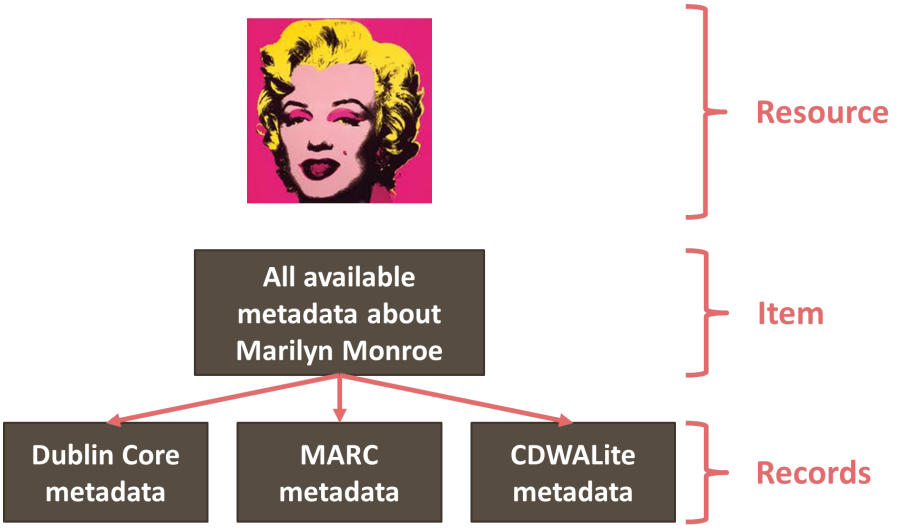

Figure 3.7: Basic Concepts of the OAI-PMH Protocol.

3.6.3 Preservation Services

A. Characterisation Service

Before ingesting the SIPs into the archive, the characterisation service is called for identifying the file formats of the linked multimedia files. This is needed for automatically generating PREMIS *Object* descriptions of those files. These PREMIS *Object* descriptions will support the migrations in a later phase. Every action performed on such a PREMIS *Object*, will get related to that *Object* and will be modelled as a PREMIS *Event*. This way, the platform is able to store and track the provenance of the descriptive metadata and the referenced multimedia files.

The reason this characterisation takes place before ingest into our archiving platform, is that one of the data streams of the SIPs has already undergone a mapping, i.e., the DC RDF records are the mapped versions of the harvested original records. For this, the SIP needs to be extended with the PREMIS *Object* instances of these metadata records. A PREMIS *Event* instance will relate the original metadata record to the mapped DC RDF record.

For this characterisation service, we employed DROID[16]. This tool is designed to automatically identify and validate file formats and to

[16] `http://droid.sourceforge.net/`

output preservation-related metadata from these files. This output is transformed into a PREMIS *Object* description. This information gets enriched with information from the *Preserv2* format registry[17]. This registry publishes the *PRONOM* database[18] as LOD, enriched with information from *DBpedia*. This enrichment gives extra information on the needed environment to render or create such file formats. This object description is also ingested into the SIPs, extending them to AIPs.

B. SIP Ingest Service

The SIPs delivered to the archive and extended to AIPs by our characterisation service, now need to be ingested into the archive. This service takes the SIP, packed in the *BagIt*[95] format, and transforms it so it can be stored into our archive. In the archive, we work file-based. This means the AIPs are actually folders on the file system of the archive. The folder these AIPs are ingested to is actually a folder equipped with distributed storage capabilities. For this *Moosefs*[19] was chosen, as distributed file system. This allows to work with traditional folders, while in the backend Moose serves the distributed storage.

Such an object folder is our actual AIP of our long-term preservation platform. RDF Fragment 3.1 gives an overview of the typical structure of our AIP / folder. This means the original metadata file, the mapped DC RDF description, the multimedia files referenced by the metadata, the transcoded multimedia files, and the generated provenance information are stored in this object folder. The mapped DC RDF description and preservation metadata is ingested in our triple store later on in the workflow. This way, our semantic preservation information (PREMIS object instances of the data streams, their linked events, rights and the related agents) will also be published as LOD.

RDF Fragment 3.1: A typical AIP in our archive: example of a object folder.

```
/[object-ID]
    origin.xml
    dc.rdf
    stories.wma
    stories.mp4
    provenance.rdf
```

[17] http://p2-registry.ecs.soton.ac.uk/
[18] http://www.nationalarchives.gov.uk/PRONOM/Default.aspx
[19] http://www.moosefs.org/

C. Transcodings

After the SIP ingest service our AIP of the harvested data is complete. It is ready to support preservation actions, e.g., validation services for virus checking, MD5 checking, digital signature checking, etc. One of these actions are thus migrations to keep the information accessible for the end-user. For implementing the migration services, we used three packages: *ImageMagick*[20], *FFmpeg*[21], and *JODConverter*[22]. *ImageMagick* is a free software suite for creating, and editing bitmap images. It supports over 100 image formats, e.g., JPEG, JPEG2000, TIFF, PDF, etc. *ImageMagick* has several program interfaces. We used its *java* interface, *JMagick*, for creating image migration services. *FFmpeg* is a free solution for converting and streaming audio and video. It supports many file formats, video, and audio codecs. It provides a command line interface, which is used for implementing its audio and video migration services. The *JODConverter* is an open source project providing a *java* library for converting various text documents. This library was employed for implementing the text migration services.

For demonstration purposes, we implemented two sorts of migrations: one for dissemination, and one for preserving an interpretable master copy. The first kind of migration can be lossy, but reaches greater compression ratios. The latter must be lossless and analysed properly, because the master copy is the highest quality, interpretable copy of the original digital file. This can be indicated in the PREMIS *Object* description by the datatype properties *preservationLevel* ('0' for the master copy, '1' for the dissemination copy) and *preservationLevelRole* ('master copy','dissemination'). Other sorts of migration services could also be implemented by an archive, e.g., making a copy for editing. This distinction allows the archive to delete from time to time older dissemination copies, which are not supported anymore. Master copies will not get deleted for preservation purposes. These migration services should be supported by the archives policies and preservation plans. These must be thoroughly analysed not to lose any quality of the master copy. These migration services also update the preservation information (provenance.rdf) stored in our AIP / distributed object folder with a PREMIS *Event* instance, describing the migration.

[20] `http://www.imagemagick.org/script/index.php`
[21] `http://ffmpeg.org/`
[22] `http://www.artofsolving.com/opensource/jodconverter`

3.6.4 Dissemination Services

Now, we have the AIP and preservation services in place. The data still needs to be disseminated if the rights allow it. For its dissemination, we publish the records, together with their provenance information as LOD. LOD refers to a style of publishing and interlinking structured data on the Web and lets you use RDF data models to publish the structured data on the Web and uses RDF links to interlink data from different datasets. Before publishing these records, they are enriched by our enrichment service. The publication of the archived content and their provenance information is discussed in detail in the next Chapter.

A. Reconciliation Service

The incoming metadata is mapped using generic XSLT mappings for the supported metadata models. Although they are standards and well documented, the metadata models are always used differently in the providing institutions. An example of this are the person names. Some institutions will use 'first name last name', while others will use 'last name, first name'. Other examples are date formats or specific properties for referencing the multimedia files. To harmonise the metadata before enriching it, a reconciliation step is performed. This step is specific for a certain data provider. The reconciliation service makes use of N3 rule files to do the providers specific reconciliation. This way, the differences between the metadata delivered by different institutions can be unified.

B. Enrichment Service

All the archived AIPs possess a DC RDF description. These descriptions, together with the provenance information stored in the AIP, are ready for dissemination. However, we advocate a further step aiming at enriching the (meta)data semantically before ingesting it into the LOD server following the Linked Data principle. With this enrichment step, we want to identify named entities into our metadata descriptions and link these named entities to other resources in the LOD cloud describing the same named entity. This way, extra information on the detected named entities can always be offered to the end user.

In our case, we apply linguistic processing on the plain text contained into the 4 W-elements (what, where, who, and when) of the metadata and the main description of the DC RDF records. The linguistic processing consists in extracting named entities such as persons, organisations, companies, brands, locations, and other events. We use the *OpenCalais* infrastructure[23] for extracting these named entities. For example, the processing of the free text description field in the DC description of an event "Tom Barman and his band dEUS opening their latest album Vantage Point in Rock Werchter" will result in five named entities: 'Tom Barman','dEUS', 'Vantage Point', 'Rock Werchter', and 'Werchter' together with their type (i.e., Person, Music Group, Music Album, Event, Location, etc.). These detected named entities become resources in the triple store. In our triple store, we already have resources, i.e., the records themselves, the collections they belong to, the institutions they belong to, the dates they happen, the places they take place, and the involved persons. All these resources are enriched with formalised knowledge on the Web available in *GeoNames*[24] for the locations, in *DBPedia*[25] for the persons, organisations and events, in *BibNet*[26] for authors, singers and music bands, and in *Toerisme Vlaanderen*[27] for touristic information on locations. This way, our approach provides *i)* unique identifiers for the resource and *ii)* formalised knowledge about this person such as his biography, career, and genealogy in multiple languages.

The main challenge in this semantic enrichment step is then to deal with the ambiguity. For example, the *GeoNames* Web service tends to return all the cities named 'Dublin' on the planet when a single string 'Dublin' is passed as an argument, and no country is specified. Fortunately, most records contain always information about the city and the country yielding accurate recognition of the location mentioned in the event. When no country is specified and the *GeoNames* service returns several results, the editor chooses the right enrichment. This ambiguity problem also comes into play for enriching persons.

[23] `http://www.opencalais.com/`
[24] `http://www.geonames.org`
[25] `http://dbpedia.org`
[26] `http://www.bibnet.be/`
[27] `http://www.toerismevlaanderen.be`

C. LOD Ingest Service

When our records are enriched by the enrichment service, these can be published by our LOD server, if the rights permit it. This LOD server publishes the content of our triple store. The institutions have the choice to preserve and publish their records or to only publish their records. If the latter is the case, the harvested, mapped and enriched records are ingested into the triple store of the LOD server using this LOD ingest service. For the triple store Openlink Virtuoso[28] was used, a high-performance persistent triple store. The publication of the harvested data and their provenance data is discussed in detail in the next chapter.

The rights, described as PREMIS *RightsStatements*, are very concise. For our platform, only two different rights statements are defined. One for only disseminating the data, and one for both dissemination and preservation of the data. These granted rights are described in the rights statements. These instances will determine if a record can be disseminated by our LOD server or not.

3.7 Conclusions

In this chapter, we have presented a distributed, digital long-term archive relying on semantic technologies. Our platform is able to harvest data, store it for the long-term, and disseminate it as LOD. This data comes from very diverse institutions, each using domain-specific metadata formats. For this, we have developed a layered, semantic metadata model. The top layer lets the archive deal with the diverse data coming from the institutions. For this layer, DC RDF was chosen. The bottom layer will enable the long-term preservation processes and consists of our semantic version of the PREMIS 3.0 data dictionary, i.e., PREMIS OWL, another standardisation work carried out in the light of this research. Using this ontology, it is possible to store the metadata needed for the preservation services. It forms the data model for the archive. A SOA was designed for this distributed archive. This SOA in combination with an ESB allows to modify and expand the current setup of processes and to communicate with all the distributed preservation and dissemination services. These service are implemented on top of the archiving server. This server has direct

[28] `http://virtuoso.openlinksw.com/`

access to a distributed file system for storing the archived object file-based and to the triple store, which stores the metadata (descriptive and preservation metadata), apart from their file-based storage. This triple store allows to provide SPARQL endpoints over the archived data and supports publication of the RDF data as Linked Open Data.

The author's work on the design and formalisation of this ontology, and, more generally, the generation of provenance information led to the following publications:

- Sam Coppens, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. „PREMIS OWL - A Semantic Long-Term Preservation Model". In: *International Journal on Digital Libraries* (2014)
- Paul Bastijns, Sam Coppens, Siska Corneillie, Patrick Hochstenbach, Erik Mannens, and Liesbeth Van Melle. „(Meta)datastandaarden voor digitale archieven". dut. In: (2009). Ed. by Rik Van de Walle and Sylvia Van Peteghem, p. 199. URL: `http://www.archive.org/details/metadatastandaardenVoorDigitaleArchieven`
- Rik Van de Walle, Sam Coppens, and Erik Mannens. „Een gelaagd semantisch metadatamodel voor langetermijnarchivering". In: *BIBLIOTHEEK-EN ARCHIEFGIDS* 84.5 (2009), pp. 17–22
- Sam Coppens, Erik Mannens, and Rik Van de Walle. „Semantic BRICKS for performing arts archives and dissemination". eng. In: ed. by Tom Evens and Dries Moreels. Access to Archives of Performing Arts Multimedia. Vlaams Theater Instituut (VTI), 2009, pp. 121–141. ISBN: 9789074351386
- Stijn Notebaert, Jan De Cock, Sam Coppens, Erik Mannens, Rik Van de Walle, Marc Jacobs, Joeri Barbarien, and Peter Schelkens. „Digital recording of performing arts: formats and conversion". eng. In: ed. by Tom Evens and Dries Moreels. Access to Archives of Performing Arts Multimedia. Vlaams Theater Instituut (VTI), 2009, pp. 95–119. ISBN: 9789074351386
- Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Sam Coppens, Erik Mannens, and Rik Van de Walle. „R&Wbase: Git for triples". In: *Proceedings of the 6th Workshop on Linked Data on the Web* (2013)
- Tom De Nies, Evelien D'heer, Sam Coppens, Davy Van Deursen, Erik Mannens, and Rik Van de Walle. „Bringing Newsworthiness into the 21st Century". In: *11th International Semantic Web Conference (ISWC-2012)*. 2012, pp. 106–117

- Tom De Nies, Sam Coppens, Davy Van Deursen, Erik Mannens, and Rik Van de Walle. „Automatic discovery of high-level provenance using semantic similarity". In: *Provenance and Annotation of Data and Processes*. Springer Berlin Heidelberg, 2012, pp. 97–110
- Sam Coppens, Erik Mannens, Raf Vandesande, and Rik Van de Walle. „Digital long-term preservation, provenance and linked open data". In: *2011 European Library Automation Group conference (ELAG 2011): It's the context, stupid!* 2011
- Erik Mannens, Sam Coppens, Toon De Pessemier, Hendrik Dacquin, Davy Van Deursen, and Rik Van de Walle. „Automatic news recommendations via profiling". In: *Proceedings of the 3rd international workshop on Automated information extraction in media production*. ACM. 2010, pp. 45–50

Chapter 4

Versioned Data on the Web - Memento datetime Content Negotiation and Provenance Publication

Publishing dynamically changing data, together with all its different versions, remains problematic on the Web, because a standardised protocol for this is missing. The more data becomes interlinked and dynamic, the more this issue forms a problem. This also holds true for Linked Open Data. In this chapter, we provide a way for publishing versioned data together with its provenance information on the Web. The W3C Provenance standard already provides a way of disseminating provenance information. This provenance information actually relates all the different versions of the data. Memento already provides a way for datetime content negotiation. Our framework actually combines both techniques to publish versioned linked data on the Web. This way, our framework allows to publish the information of a resource as Linked Open Data, including all its previous versions and their provenance information, in a web-accessible manner.

4.1 Introduction

At the moment a lot of the information in the Linked Open Data cloud are static data dumps. Publishing static data as Linked Open Data forms no problem. However, these datasets are becoming more and more dynamic. This can be seen with the uprising Web of Things,

where even sensory data is being published on the Web. Recently, W3C started a standardization activity to create a Linked Data Platform, with as main goal to define a RESTful way to read and write Linked Data [33]. The consequence of this is that published datasets will become even more dynamic.

With this upcoming dynamic Linked Open Data a new problem arises concerning the interlinking of these dynamic resources. For instance, some enrichments of the data being published as LOD can become invalid over time. To give an example: a record on the second Gulf War is interlinked with a record of the president of the United States, George W. Bush at that time. Today, this enrichment would refer to the current president, i.e., Barack Obama, which makes the enrichment invalid.

In this chapter, we introduce a way for publishing dynamic, i.e., versioned, data as Linked Open Data on the Web, such that even the enrichments of the published data remain valid over time. At the same time, we are able to publish the provenance information of the different versions of the data too. Eventually, this provenance information is relating all the different versions of a piece of data. Our way of publishing this versioned linked data on the Web is, of course, completely compliant with the Provenance standard of W3C for publishing provenance information on the Web.

Our idea is demonstrated using the long-term preservation platform, explained in Chapter 3. Our developed platform generates many different versions of the harvested data, i.e., metadata and referenced multimedia files, via its preservation processes. These resources, their previous versions and their provenance information, relating the different versions, will be published on the Web as LOD. When preserving information for the long-term and publishing the information as LOD at the same time, different problems arise. First of all, we need to have persistent URIs for our resources, which will publish the information of a certain version of the resource. Another problem involves the enrichments that occur on the resources before publishing them as LOD. These enrichments will not always remain valid over time. We need a way for preserving the temporality of these enrichments. The last problem being tackled in this paper is the publication of the provenance information on the Web which will allow automatic discovery of the provenance information.

To solve these problems, our developed platform is extended with the Memento[1] [96] datetime content negotiation. This datetime content negotiation will allow to select the appropriate version, called memento in the Memento framework, of the archived information and to publish it on a persistent URI. This datetime content negotiation will also solve the problem of preserving the temporality of the enrichments of the archived information. The different versions of the archived information are linked to each other via their provenance information. To publish the provenance information of each version on the Web, we extended the Memento framework to offer provenance links using a special Hypertext Transfer Protocol (HTTP)[12] link header for automatic discovery of the provenance information.

In this chapter, we present how our digital long-term preservation platform is able to publish the provenance information on the Web. First, Section 4.2 describes some related work on this topic. Then, in Section 4.3, we explain in short our semantic layered metadata model, which allows the archive to deal with the diversity of metadata records coming from diverse institutions and to track the provenance of the harvested data. Section 4.4 describes the distributed architecture of the archive and its processes and how they produce different versions of the data. Section 4.5 explains the publication of the content and its provenance information using the Memento framework, extended to provide provenance information. We end with a conclusion in Section 4.6.

4.2 Related Work

Interest in digital preservation can be seen by the multitude of projects in this area. Planets (Preservation and Long-term Access through Networked Services)[2] was especially aimed at defining guidelines for preservation planning. However, it did not tackle the integration of different existing metadata formats, or the dissemination of the metadata as LOD. Likewise, the Prestospace (Preservation towards storage and access) project's objective was to provide technical solutions and integrated systems for a complete digital preservation of all kinds of audio-visual collections [3]. The project was especially focussed on the underlying technologies, e.g., automated generation of metadata or

[1] http://www.mementoweb.org

[2] http://www.planets-project.eu/

[3] http://prestospace.org/project/index.nl.html

detection of errors in content [84], but without using a standardised, semantic preservation model to support the archiving, nor did they tackle the problem of publishing the generated provenance information on the Web.

The CASPAR project (Cultural Artistic and Scientific knowledge for Preservation, Access, and Retrieval) presented technologies for digital preservation [4]. The OAIS Reference Model was chosen as the base platform, and the project was focused on implementing the different steps in the preservation workflow. They focus more on preservation services than on describing the preservation information. BOM Vlaanderen [5], a national research project, was aimed at preservation and disclosure of audio-visual content in Flanders. Additionally, it looked at ways to unify different metadata standards currently used for describing audio-visual content. Current trends are on integrating different media archives. PrestoPRIME researches and develops practical solutions for the long-term preservation of digital media objects, programmes and collections, and finds ways to increase access by integrating the media archives with European on-line digital libraries in a digital preservation framework [6].

The previous discussed related work was focusing on the digital long-term preservation, not on the more general problem of enabling their provenance information on the Web. For the work done is this area, the work of the W3C Provenance Working Group[7] is the major reference. This working group produced a data model for provenance information, provided several formaisations of the data model, and foresees a way for publishing this provenance information on the Web. In our framework, we publish the provenance information as Linked Open Data using PREMIS OWL, explained in Chapter 2. This information is only interoperable in the long-term preservation context, where PREMIS is well-known, not in a Web context, where PROV-O, the provenance ontology, offered by the Provenance Working Group, should be used. Future work could involve publishing the provenance information using this vocabulary, which is more suited for publication on the Web than PREMIS OWL, which is intended to be a data model for digital long-term archives.

[4] http://www.casparpreserves.eu/
[5] https://projects.ibbt.be/bom-vl
[6] http://www.prestoprime.org/
[7] http://www.w3.org/2011/prov/

Another interesting work done in the area of publishing provenance for linked data is the paper of Olaf Hartig and Jun Zhao published at IPAW [88]. In that paper they describe the Provenance Vocabulary[8] used for describing the provenance information as Linked Open Data. Next to this, they also offer ways of publishing this provenance information for Linked Data. They discuss how provenance can be added to Linked Data objects, how provenance can be included into RDF dumps and how the provenance information can be queried using SPARQL endpoints. This work enables provenance for Linked Data, but it does not offer solutions for automatic discovery of the provenance information or ways for publishing provenance on the Web beyond using semantic web technologies.

4.3 Layered Metadata Model

In this section, I will re-introduce shortly the layered metadata model of our long-term preservation infrastructure, as explained in the previous chapter. When publishing different versions of a resource, this resource needs a persistent identifier. The different versions of a resource will get a URI of their own, linked to the persistent identifier. The layered data model will define which resources get a persistent identifier.

Inside a digital archive, the unit of information the archive works with is packaged into a SIP, i.e., the Submission Information Package. The SIP contains descriptive metadata together with the preservation metadata. The descriptive metadata serves to support the management of the resources to be preserved. The preservation metadata is responsible for supporting the preservation services for that given resource. For the descriptive metadata layer, Dublin Core RDF [89] is used. The preservation metadata layer is described using PREMIS OWL, as discussed in Chapter 2.

If we take back Figure 2.3, we showed how a record is actually conceived in PREMIS OWL. This model will define our persistent identifier for our dynamic records, as shown in Figure 4.1. Thus, a record is actually aggregated into a SIP package. This package contains the metadata and the referenced multimedia files. As explained by Figure 2.3, such a SIP package is in PREMIS OWL actually a *premisowl:IntellectualEntity* instance, with its related *premisowl:Object* instances. The persistent identifier of our SIP package (or record) is in fact the identifier of this *premisowl:IntellectualEntity* instance.

[8] `http://purl.org/net/provenance/`

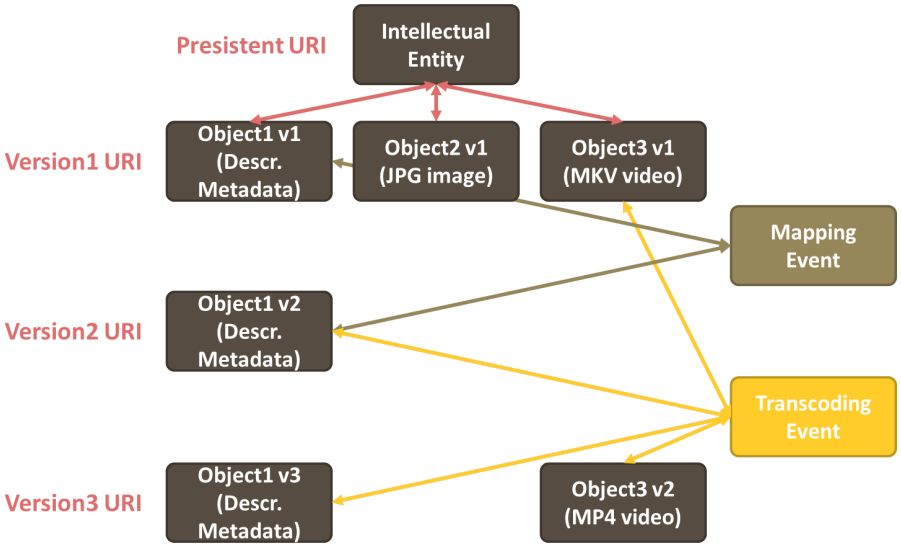

Figure 4.1: Schematic Overview of the Record and its different Versions.

The different versions of the record are in PREMIS OWL reflected by the different *premisowl:Object* instances of the metadata record. Changes to the record leading to a new version can be descriptive metadata changes, e.g., mappings or reconciliations, and changes in the referenced files in the descriptive metadata, e.g., video files which are transcoded for dissemination on the Web. Whenever a referenced file changes, its descriptive metadata will, of course, change accordingly to refer to the newly created file. Hence, the *premisowl:Object* instances of the descriptive metadata reflect all the versions of the archived record.

4.4 Architecture

In this section, our architecture of the digital long-term preservation archive is briefly described. This architecture will give insight in the different versions that are created and their provenance information. For an in-depth description of the architecture, we refer to Chapter 3. Our platform has a service oriented architecture[9] (SOA). This SOA will make use of a central service hub, which will offer the needed services for the platform.

[9] `http://opengroup.org/projects/soa/`

The objectives of our platform are twofold:

- Disseminate the content and provenance information as LOD.
- Enable long-term preservation.

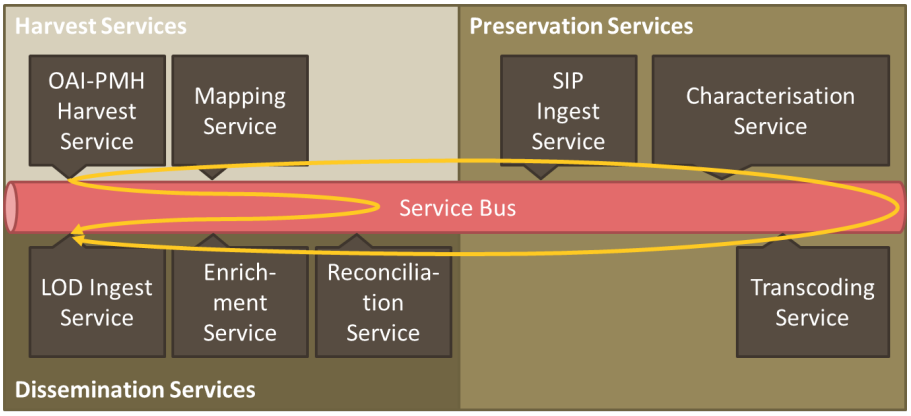

Figure 4.2: Schematic Overview of the Service Bus and its Connected Services.

To support these two functionalities, we have designed two workflows, i.e., a dissemination workflow and a preservation workflow, as depicted by Figure 4.2. The whole preservation/dissemination cycle starts with a **harvesting process**, which will harvest the metadata, and the referenced files. This will typically form the first version of the record to be preserved, namely its original content. The metadata harvested, is described using several descriptive metadata formats, e.g., MARC, DC, or CDWA. For management and dissemination purposes this metadata needs to be mapped to DC RDF. For this, we rely on a **mapping service**, which will map the incoming metadata to DC descriptions. This mapping service will create our second version of the record.

If the content also needs to be preserved, the original metadata record, the mapped DC RDF record and the referenced files get packed into a Submission Information Package (SIP), according to the OAIS specifications by the **SIP creator service**. For this SIP, the *BagIt* [95] package format is used. This SIP package is then ingested into the archive, using the **SIP ingest service**.

When ingesting this *BagIt* package into the archive, it has to be supplemented with the preservation information to form an Archival Information Package (AIP) in the OAIS terminology. This package holds all the different versions of the metadata and the multimedia files, referenced by the metadata files. For this preservation information, we will use our PREMIS OWL ontology. During this ingest process, all files in the package get a PREMIS *Object* description, related to the mapped DC RDF description, thus becoming the PREMIS *intellectual entity*. For this we rely on a **characterisation service**, which will identify the file format of the files and model the files as PREMIS *Objects*. Every action performed on such a PREMIS *Object*, will get related to that *Object* and will be modeled as a PREMIS *Event*. This way, the platform is able to store and track the provenance of the descriptive metadata and the referenced multimedia files.

The next thing within the workflow is the migration of the stored, related multimedia files. These files get migrated to a certain file format, defined by the archives' preservation plans. Such a preservation plan can stipulate, e.g., that all image files must be migrated to the TIFF file format to keep the image information accessible for long-term preservation purposes, or, e.g., that all image files must be migrated to the JPEG file format to keep the image information accessible for dissemination purposes. For this, we need **transcoding services**, which can then migrate various incoming file formats to the appropriate file format according the preservation plans. This migration will extend the AIP package with the extra migrated data stream. This data stream is then passed to the characterisation service to get a PREMIS *Object* description of the generated data stream and the preservation information is also extended with a description of the migration service as a PREMIS *Event* relating the source object to the migrated object. This transcoding service will create new versions of the referenced multimedia files. These newly created multimedia files will be referenced by the descriptive metadata. Thus, this service creates actually a new version of the record to be preserved, and this for each referenced multimedia file, until all multimedia files are transcoded to the file formats, defined by the preservation plan.

During the last phase, the archived information is moved to the LOD server for dissemination of the information. For this, the descriptive DC RDF metadata will get reconciled by the **reconciliation service**, and then enriched by the **enrichment service** before it gets ingested into

the LOD server's triple store by the **LOD ingest service**. Both services create new versions of the descriptive metadata. The reconciliation is different for each data provider and is defined by a rule file. This will, for instance, give all names the same format, e.g., first name last name, to support the enrichment. For the enrichment service, the platform relies on data sources like the *OpenCalais* infrastructure[10] for extracting these named entities, *GeoNames*[11] for enriching the locations, *DBPedia*[12] for enriching the persons, organisations and events, *BibNet*[13] for authors, singers and music bands enrichment, and *Toerisme Vlaanderen*[14] for touristic information enrichment on locations. This way, our approach provides *i)* unique identifiers for the resource and *ii)* formalised knowledge about this resource.We will not only disseminate the intellectual entity, i.e., the descriptive metadata, but also the preservation information, so the end-user has access to all the information available about that object.

If the harvested content does not need to be preserved, it is directly routed to our **enrichment service**, which will interlink the data with external data sources after harvesting and mapping the metadata. This enriched DC description then gets ingested into the triple store of the LOD server, which automatically publishes the enriched DC records as LOD.

4.5 Publication

Our architecture, described in the previous section, ingests all the harvested and generated information into our triple store. This information, including the provenance information, needs to be disseminated as Linked Open Data. For this dissemination, we want to have stable URIs [11], e.g., http://../record/PresidentOfTheUSA for the harvested original resources. These resources change over time via the preservation processes. As such, every version of the resource has another URI, e.g., http://../record/PresidentOfTheUSA_V3. To link from the original resource with a stable URI to the appropriate version URI, we extended our Linked Open Data server with the Memento datetime content negotiation[15], besides the mediatype content negotiation. This

[10] `http://www.opencalais.com/`
[11] `http://www.geonames.org`
[12] `http://dbpedia.org`
[13] `http://www.bibnet.be/`
[14] `http://www.toerismevlaanderen.be`
[15] `http://datatracker.ietf.org/doc/draft-vandesompel-memento/`

mechanism allows the platform to publish the information on persistent URIs. Based on the Memento datetime content negotiation the right version of that resource is selected and published as LOD. This mechanism is depicted in Figure 4.3 and explained in publication [97].

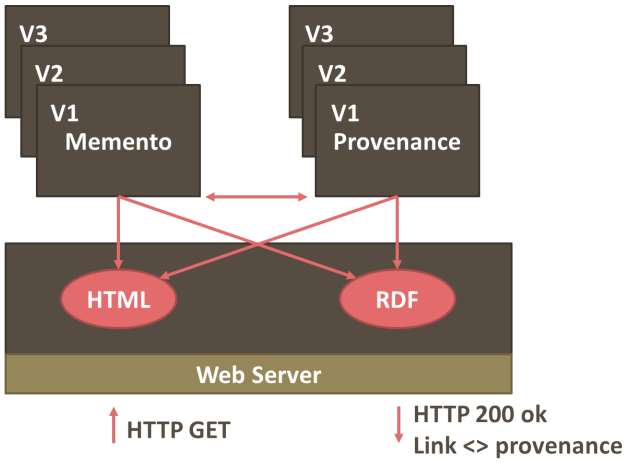

Figure 4.3: Schematic Overview of the Content Negotiation.

4.5.1 Memento Datetime Content Negotiation

The Memento framework is based on HTTP and HTTPS URIs and introduces several concepts:

- Original Resource (URI-R): This resource is archived for the long-term and has several versions.
- Memento (URI-Mj): This resource refers to one of the versions of an original resource.
- TimeGate (URI-G): The TimeGate for an original resource is a resource that supports the datetime content negotiation.
- TimeMap (URI-T): A TimeMap for an original resource lists the URIs of all the mementos of that original resource.

The Memento framework is based on HTTP request and response headers. The framework introduces two new headers: Accept-Datetime and Memento-Datetime. The Accept-Datetime header is used to ask for the version of the original resource valid on that time. If a user agent requests an original resource for a specific datetime, the server responds with a link to the timegate, which can do the datetime content negotiation for that original resource. The timegate redirects the user

agent to the appropriate memento, which responds with a memento-datetime. This memento-datetime gives the datetime the resource was created. This datetime of a memento is retrieved using the provenance information of that memento. The provenance of every memento is modeled as a PREMIS OWL Object instance relating to Event instances. Such an Object instance has always a creation event. The datetime of this creation event is used for the Memento datetime content negotiation. RDF Fragment 4.1 gives an example of such an HTTP interaction.

```
1: UA ——— HTTP GET/HEAD; Accept-Datetime:Tj ————————> URI-R
2: UA <——— HTTP 200; Link:URI-G ———————————————— URI-R
3: UA ——— HTTP GET/HEAD; Accept-Datetime:Tj ————————> URI-G
4: UA <——— HTTP 302; Location:URI-Mj; Vary;
                     Link:URI-R,URI-T,URI-Mj ————————— URI-G
5: UA ——— HTTP GET URI-Mj; Accept-Datetime:Tj ————————> URI-Mj
6: UA <——— HTTP 200; Memento-Datetime:Tj;
                     Link:URI-R,URI-T,URI-G,URI-Mj ——— URI-Mj
```

RDF Fragment 4.1: Typical Memento HTTP Interaction

Next to the two new headers, Memento also introduces some new values for the existing HTTP headers: Vary and Link. The value for the VARY header in our case will be *negotiate, accept-datetime, accept*. This VARY header informs that the content negotiation was performed in two dimensions, i.e., the datetime content negotiation and the media type content negotiation. The relation types for the Link header Memento introduced are *original*, for referencing the original resource, *timegate*, for indicating the timegate, *timemap* for linking to the timemap, and *memento* for referencing to various mementos for an original resource. These Link headers allow automatic discovery of the timegate, the timemap, the original resource and several other mementos.

Introducing this Memento datetime content negotiation is justified from our digital long-term preservation perspective. A problem we were facing publishing information as Linked Open Data and preserving it at the same time, involved the enrichments. These enrichments do not always remain valid over time. That is why these enrichments are mostly left out of the metadata to be stored for the long-term. If the data providers of the enrichments also support the datetime content negotiation, a memento with enrichments would reference that memento of the enrichment when it was valid. In other words, the Memento datetime content negotiation also preserves the temporality of the information. This also justifies storing the enrichments of the metadata records for the long-term.

4.5.2 Publishing Provenance

In our platform, every version (memento) of a harvested resource (original resource) has a PREMIS OWL Object description. This Object description describes the provenance of that object and is related through events to object descriptions of other versions/mementos of that original resource. This allows our platform to include in the response of the request for a memento a *provenance* link header which includes the link to the LOD published PREMIS OWL Object description (*URI-Pj*) of that memento. This *provenance* link header will allow automatic discovery of the provenance information. We extended the Memento framework with a new concept:

- Provenance (URI-Pj): This resource refers to the provenance of the selected version/memento of the original resource.

To allow this resource to be automatically discovered, we extended the Memento framework with a special value for the existing HTTP header Link referencing the provenance information. As HTTP header Link, the standardised HTTP header link for provenance is used [98]. The relation type for this Link header is <http://www.w3.org/ns/prov#has_provenance> for the current provenance record (URI-Pj). The anchor of the HTTP header Link refers to the resource the provenance is given for in this header Link. If no anchor parameter is given, then the current URI is assumed to be the resource provenance is provided for. This way, our extended Memento framework is completely inline with the provenance standard of W3C. A typical HTTP interaction, requesting a certain memento, is shown in RDF Fragment 4.2.

The provenance records are themselves also datetime content negotiable. So they become mementos of an original provenance resource. Doing this, gives some extra benefits. The Memento framework defined some extra relation types for the HTTP Link header referencing a memento. When applied to a provenance record of a memento of an original resource, they get the following definitions:

- first memento (URI-M0): This resource refers to the provenance of the first version/memento of the original resource.
- last memento (URI-Mn): This resource refers to the provenance of the last version/memento of the original resource.
- memento (URI-Mj): This resource refers to the provenance of the selected version/memento of the original resource.

- previous memento (URI-Mi): This resource refers to the provenance of the previous version/memento of the selected version/memento of the original resource.
- next memento (URI-Mk): This resource refers to the provenance of the next version/memento of the selected version/memento of the original resource.
- timemap (URI-T): A TimeMap for a provenance record of an original resource lists the URIs of the provenance records of all mementos of that original resource.

The response for a memento request will include a provenance header link, referencing the provenance information of that memento. This provenance record is on itself also a memento. The response of this memento includes a *timemap* link header pointing to a URI (*URI-T*) listing the URIs of the provenance records of all mementos of that original resource. This way, an agent can immediately have an overall view on the provenance of an original resource.

These extra links could be very helpful in processing the provenance information. Our PREMIS OWL model allows describing digital signatures, signing the versions/mementos of that original resource. A quality checker could investigate the quality and trustworthiness of the published information. This quality checker could investigate the digital signature of the last version. If this was signed by a trusted party and the digital signature is still valid, the quality checker could immediately move on to the provenance of the first memento to check where the signed information came from. The quality checker can then check if that data provider is also a trusted party to make an overall judgement regarding the quality and trustworthiness of the information. The PREMIS OWL model also allows describing the rights information in the provenance of a resource, such as licenses, copyrights, rights granted, etc. A license checker could use these additional links to browse through the provenance records of the mementos of an original resource and check if none of them violates the rights information of another memento.

A shortcoming of making provenance records also datetime content negotiable, is that all events happening on a preserved resource more recent than the datetime asked for, will be left out of the provenance description. Hence, the provenance information would then only contain links to older versions/mementos of the preserved resource and the links to the more recent versions are lost.

```
1: UA ——  HTTP GET/HEAD; Accept-Datetime: Tj ——————> URI-R
2: UA <—— HTTP 200; Link: URI-G ———————————————— URI-R
3: UA ——  HTTP GET/HEAD; Accept-Datetime: Tj ——————> URI-G
4: UA <—— HTTP 302; Location: URI-Mj; Vary; Link:
      URI-R, URI-T, URI-Mj,————————————————————— URI-G
5: UA ——  HTTP GET URI-Mj; Accept-Datetime: Tj ———> URI-Mj
6: UA <—— HTTP 200; Memento-Datetime: Tj; Link:
      URI-R, URI-T, URI-G, URI-Mj, URI-Pj ————————— URI-Mj
```

RDF Fragment 4.2: Extended Memento HTTP Interaction with Provenance Information

To improve the automatic discovery of the provenance information of a memento, our platform will inject the provenance link of the memento also in the HTML and RDF descriptions of that memento. This will enhance the provenance discovery, because not all clients will be able to intercept the new *provenance* link header. For the HTML representation of the memento, our framework includes a HTML link tag in the head of the HTML document. This link has a relation type of http://www.w3.org/ns/prov#has_provenance, e.g., <link rel="http://www.w3.org/ns/prov#has_provenance" href="http://../object/PresidentOfUSA_V3"/>. The resource's provenance is provided by a second link element, i.e., http://www.w3.org/ns/prov#has_anchor, e.g., <link rel="http://www.w3.org/ns/prov#has_anchor" href="http://../record/PresidentOfUSA_V3"/>. These HTML head links are actually recommended by the provenance access and query specification of the W3C provenance working group, keeping our extension compatible with that provenance specification. For the RDF representation, our platform injects a triple denoting the provenance information of that memento. For linking this provenance record (PREMIS OWL Object instance), the PREMIS OWL object property *hasLinkingObject* is used. An example of such an injected triple in the RDF description of a memento is: <http://../record/PresidentOfUSA_V3> premis:hasLinkingObject <http://../object/PresidentOfUSA_V3>. This property is equivalent to the standardised property for referencing provenance by Prov-O [99], the OWL formalisation of the provenance data model, provided by the W3C provenance working group.

In some cases, it might be convenient to store the provenance of the provenance information. An example of this in our framework is the characterisation process. This process identifies a memento of an

original resource and creates a PREMIS OWL Object instance of it. This can be the metadata record or a multimedia file referenced in a metadata record. In case of a file, the Object description is being enriched with information from the *Preserv2* format registry[16]. This is an enrichment event occurring on provenance information. This could be described in the provenance of the provenance information. Another example of this are digital signatures. Our PREMIS OWL model allows describing these digital signatures applied to a stored memento, but digital signatures can also be used to sign provenance information. When including a *provenance* Link header in the response to a provenance record, the provenance of the provenance information can be discovered.

Looking at the 5-star deployment scheme[17] of Tim Berners-Lee, this framework could add two more stars for indicating the rating of a Linked Open Data provider. A sixth star could go to Linked Open Data providers that support the Memento datetime content negotiation. This sixth star will indicate to, e.g., a long-term preservation archive, that the enrichments coming from that provider could be stored also for the long-term, as discussed earlier. A seventh star could go to Linked Open Data providers not only supporting the Memento datetime content negotiation, but also using this framework to publish their provenance records as Linked Open Data. This seventh star will indicate that the data provider publishes provenance information and, hence, it is possible to make trust judgments over that data using quality checkers or license checkers, as mentioned above.

4.5.3 Implementation

For implementing this framework, we used Jena TDB as triplestore for the back-end. This is a large-scale persistent triplestore which supports SPARQL. On top of this triplestore, the LOD server was built using Apache Tomcat as HTTP web server. This LOD server has a servlet which will do the datetime and the mediatype content negotiation and will redirect from the original resource, published on a persistent URI, to the appropriate version/memento of that original resource. This servlet will form the timegate. Next to this, we have servlets to serve the appropriate mediatype of the information (HTML and RDF) and insert the provenance information. The resources that will be published with

[16] `http://p2-registry.ecs.soton.ac.uk/`
[17] `http://www.w3.org/DesignIssues/LinkedData.html`

this timegate are the harvested collections and records. As explained in the previous section, we do not offer datetime content negotiation for the provenance information. For this information, we have a separate servlet only supporting media type content negotiation.

Next to the LOD server supporting the datetime content negotiation, we have an integration server which will provide the needed preservation processes. These preservation processes will generate the different versions of the harvested information. For the integration server we used the *Porthus .NET* Integration server, as discussed in Chapter 3.

4.6 Conclusions

In this chapter, we have tackled the problem of publishing different versions of linked data resource on the Web to a persistent identifier using datetime content negotiation. Linked data is enriched with external information. These enrichments are often timely, because both the enriched resource description as the enrichments can change over time. The presented publication mechanism for versioned linked data solves this issue. At the same time, we present a way of publishing provenance on the Web, conforming the W3C provenance standard.

To publish these different versions of a preserved resource and their provenance information, our platform relies on the Memento datetime content negotiation. We extended this framework to also include HTTP *provenance* header links for automated discovery of the provenance information. This approach allows us to disseminate the versioned information of the preserved resources on persistent URIs, depending on the datetime content negotiation to redirect to the appropriate version/memento of the original stored resource. Combining datetime content negotiation with the publication of the provenance information, links the provenance information to the datetime dimension of a certain stored resource. Finally, the framework allows discovering the provenance information of the other existing versions of an original resource bringing provenance information to the Web.

The publication mechanism is implemented on top of our long-term preservation platform, discussed in Chapter 3. This platform harvests data, stores it for the long term, and disseminates all its generated versions of the harvested resources as LOD. The basic steps supporting the dissemination in our platform are: harvesting (storing the original metadata), mapping to a common metadata representation using DC

RDF (generating a first version of the original metadata), reconciliation and cleaning up the mapped metadata (generating a second version), and enrichment of the reconciled metadata (generating a third version). Thus, this platform produces lots of different versions of the stored information and also produces provenance information, which will relate the different versions of the stored information. In our platform all the generated versions are linked to each other via provenance information, described using PREMIS OWL. With this publication mechanism we can publish all generated versions as LOD to persistent identifiers, publish its provenance information. This publication mechanism allows to even store the enrichments of the LOD published and preserved resources, because the temporality of these enrichments is also preserved during publication.

The author's work on disseminating provenance information using datetime content negotiation, and, more generally, the publication of provenance information led to the following publications:

- Yolanda Gil, James Cheney, Paul Groth, Olaf Hartig, Simon Miles, Luc Moreau, Paulo Pinheiro Da Silva, Sam Coppens, Daniel Garijo, JM Gomez, et al. „Provenance XG Final Report". In: (Dec. 2010). URL: http://www.w3.org/2005/Incubator/prov/XGR-prov/
- Miel Vander Sande, Sam Coppens, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. „Adding Time to Linked Data: A Generic Memento proxy through PROV". in: *Poster and Demo Proceedings of the 12th International Semantic Web Conference* (2013)
- Tom De Nies, Sara Magliacane, Ruben Verborgh, Sam Coppens, Paul Groth, Erik Mannens, and Rik Van de Walle. „Git2PROV: Exposing Version Control System Content as W3C PROV". in: *Poster and Demo Proceedings of the 12th International Semantic Web Conference* (2013)
- Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Sam Coppens, Erik Mannens, and Rik Van de Walle. „R&Wbase: Git for triples". In: *Proceedings of the 6th Workshop on Linked Data on the Web* (2013)
- Sam Coppens, Ruben Verborgh, Miel Vander Sande, Davy Van Deursen, Erik Mannens, and Rik Van de Walle. „A truly Read-Write Web for machines as the next-generation Web?" In: *Proceedings of the SW2012 workshop: What will the Semantic Web look like 10years from now* (2012)

- Tom De Nies, Evelien Dh́eer, Sam Coppens, Davy Van Deursen, Erik Mannens, and Rik Van de Walle. „Bringing Newsworthiness into the 21st Century". In: *11th International Semantic Web Conference (ISWC-2012)*. 2012, pp. 106–117
- Sam Coppens, Erik Mannens, Davy Van Deursen, Patrick Hochstenbach, Bart Janssens, and Rik Van de Walle. „Publishing provenance information on the web using the Memento datetime content negotiation". In: *WWW2011 workshop on Linked Data on the Web (LDOW 2011)*. Vol. 813. 2011, pp. 6–15
- Sam Coppens, Erik Mannens, Raf Vandesande, and Rik Van de Walle. „Digital long-term preservation, provenance and linked open data". In: *2011 European Library Automation Group conference (ELAG 2011): It's the context, stupid!* 2011

Chapter 5

Querying Distributed Linked Data - Query Federation using the Provenance of owl:sameAs Links

In this chapter, we focus on the use of provenance information, more specifically the use of provenance information to support query federation. Linked Open Data envisioned to create a global data space, where information is easily accessed and queried. Querying the Linked Open Data cloud as a whole still remains problematic. Prior knowledge is required to federate the queries to the appropriate datasets: each dataset provides its own SPARQL endpoint to query that dataset, and each dataset uses its own vocabulary to describe their information. In this chapter, we propose a federated SPARQL framework to query the global data space, relying on query federation and vocabulary mappings. Our query federation will traverse the owl:sameAs links. Our framework exploits the fact that owl:sameAs relationships are symmetrical and transitive. This means that such linked resources are interchangeable and form the basis for the query rewriting. The provenance of the generated owl:sameAs links is processed to resolve the SPARQL endpoints of linked datasets and their vocabulary mappings to support the query rewriting.

5.1 Introduction

The main goal of Linked Open Data is to create a global data space, the Linked Open Data cloud[1] [14], where end users can easily discover and consume data. A first key ingredient for Linked Open Data is to publish machine-readable, structured data on the Web. This allows machines to discover the data, interpret the data and act upon the data. Another key ingredient for Linked Open Data is link generation. Link generation will discover relationships between the resources being published and the resources already being described in the Linked Open Data cloud using the SPARQL protocol[25] . This step will link the data being published with information of other data sources, turning the disparate set of data sources into one global data space. Bizer et al. [100] explain very clearly the concept and technical principles of Linked Data and why link discovery between datasets is crucial in creating a global data space.

Many of these generated links will relate resources that describe the same thing. The resources describing the same thing will be linked using the *owl:sameAs* link. This link generation can be done automatically. Link generation, or enrichment, has been investigated a lot and resulted in various lookup services, e.g., DBpedia [101], GeoNames[2], or Sindice [102], and link discovery frameworks, e.g., SILK [103].

At the moment, querying this global data space as a whole remains a challenge. If we want to get results from a query that ranges over multiple data sources, accessible through different SPARQL endpoints, we will have to split up the query into sub-queries for each SPARQL endpoint and combine the results or we will have to fetch firstly all the different data sources into one data source and then query this aggregated data source. Both approaches have some limitations though:

- In the first approach, we will have to know which part of the query is meant for which SPARQL endpoint and we will need to know for each SPARQL endpoint which vocabulary it uses, so that appropriate mappings to the query can be applied.
- In the second approach, we do not need to know which SPARQL endpoints must be queried for receiving an appropriate answer, but we still need to know which vocabulary to use in the query so

[1] `http://linkeddata.org/`
[2] `http://www.geonames.org/`

that the aggregated data sources can understand the query. This approach has the danger that in a highly interlinked environment, a lot of information has to be indexed first.

This needed information for query federation, i.e., the SPARQL endpoints and vocabulary mappings, can be extracted from the provenance information of the generated links. As we will discuss later, the provenance information of these generated *owl:sameAs* links can be expressed as queries or rules. This rule gives us information on which SPARQL endpoint was contacted to find a resource describing the same thing and which vocabularies were used for querying that SPARQL endpoint. When link discovery frameworks are used for the link generation, this provenance information can be extracted from the framework's configuration. If the interlinking is done without such a framework, the queries fired for the link generation can be used as provenance of the results they generate, which are in fact the *owl:sameAs* links too.

This chapter proposes a solution to the problem of querying distributed data sources. In our approach, we will traverse the *owl:sameAs* links of the resources combined with the provenance of these *owl:sameAs* links so we know how to distribute the query to find more information on resources. The basic principle of our query federation framework is that *owl:sameAs* is a symmetrical and transitive property. This means that resources linked via *owl:sameAs* are interchangeable. This forms the basis for the query distribution algorithm, such that all the information of a resource and all its *owl:sameAs* equivalents, distributed on the Web, becomes available at a single SPARQL endpoint and creates the impression that all the information of the *owl:sameAs* equivalents is also present in the local dataset. At the same time, mappings will be applied to the query distributions to support the external dataset's vocabulary. This mapping information is retrieved implicitly by processing the provenance of the *owl:sameAs* links, as will be shown in Section 3. Because of this feature, the mappings are always defined in terms of the local vocabulary. Thus, the local vocabulary forms the universal data model for information federation. In the end, this mapping strategy is something in between a point-to-point mapping strategy and a strategy using a universal data model.

Following these *owl:sameAs* links from the Linked Open Data cloud has several benefits. First of all, Following the *owl:sameAs* links creates your own view on the Linked Open Data cloud. When publishing your information as Linked Open Data, you will only link with those

resources from datasets you trust. By following the *owl:sameAs* links for query federation, our query distribution framework only takes into account those datasets you trust and linked with. A second major advantage of following the *owl:sameAs* links is that they provide an easy mechanism to do distributed joins. In RDF all resources are identified by a URI. The *owl:sameAs* links will link two related resources to each other by means of interrelating their URIs through *owl:sameAs*. This means, the identifier of the remote resource, linked to the local resource is known and can be used to perform the distributed join operation.

The Chapter is structured as follows: in the next section, we give an overview of existing related work. Section 5.3 will elaborate on our solution, where we also discuss in detail the two main components of our framework, i.e., the *index builder* in Section 5.4, and the *query distributor* in Section 5.5. In Section 5.6, we give an overview of the optimisations that are applied to enhance the query execution performance. We end the Chapter with an evaluation and a conclusion.

5.2 Related Work

At this moment, there are several approaches to distributed querying. Some approaches rely on prior crawling and caching of the data, e.g., Sindice [102], which crawls Web pages embedding RDF and microformats and makes the crawled data available through a SPARQL endpoint and through an API. Actually, this is not distributed querying, but querying a Semantic Web index, built from crawling and caching. Another approach to distributed querying is relying on run-time link traversal to answer queries. This approach is followed by SQUIN [104]. Here, the index is built at query run-time, which avoids syncing problems. A third approach being used, is based on query federation, which is followed by frameworks like FedX: a federation layer for distributed query processing on Linked Open Data[105] and DARQ [106]. FedX provides a query distribution layer on top of the Sesame[3] framework. It executes query federation and query optimisation. DARQ federates the queries using the predicates to decide where to send triple patterns to as an optimisation technique. Just like FedX, Splendid [107] is an extension to Sesame, which employs VoID [108] to distribute its incoming queries. Splendid will start firing SPARQL ASK queries to each dataset for verification and later on statistical information is used to optimise these federated queries.

[3] `http://www.openrdf.org/`

Our framework is a combination of the link traversal approach and the query federation approach to solve queries ranging multiple, disparate datasets. It is similar to what is being done in 'Data summaries for on-demand queries over linked data [109]. This latter will use an index structure for optimising the query distribution and then queries the data real-time, so no synchronisation problems occur. Our framework will federate queries such that they follow the *owl:sameAs* links. Our approach actually exploits the fact that resources linked via an *owl:sameAs* link, are interchangeable, which automatically brings in the distributed information of these resources. At the same time, we will introduce property mappings and class mappings to solve our incoming queries and overcome interoperability issues between the linked datasets. The benefit of this approach is that a data provider gets more control over his data. He decides which datasets are used to enrich his data and, as a consequence, he controls the SPARQL endpoints to which incoming queries are distributed to.

Our framework is integrated into ARQ[4]. ARQ is a query engine for Jena[5] that supports the SPARQL RDF Query language. Many SPARQL endpoint implementations are based on ARQ. This allows for any SPARQL endpoint service provider relying on the ARQ library to easily put up its own distributed SPARQL endpoint. They just have to replace the ARQ library with our extended ARQ library and feed ARQ with the SPARQL construct queries, used for enriching their dataset, or with SILK configuration files, if they used SILK to enrich their dataset. Thus, it becomes very easy for a data publisher to set-up a distributed SPARQL endpoint, which federates queries to those datasets it is linked with.

5.3 Solution

Our distribution framework will only follow *owl:sameAs* links. The reason for this is threefold:

- The combination of all information pointed to by owl:sameAs links creates the data provider's view on the Linked Data cloud. The data provider will only link its resources to resources from data providers it trusts. By following the *owl:sameAs* links our query federation framework takes this view on the LOD cloud into account when distributing the queries.

[4] `http://jena.apache.org/documentation/query/`
[5] `http://jena.apache.org/`

- The provenance of these *owl:sameAs* links provide useful information for distributing queries. The data providers actually have this information as they are also responsible for the link generation, even when frameworks like SILK were used for the link generation.
- These *owl:sameAs* links are a way for performing distributed joins over disparate datasets. Without these links, the query distribution framework should do the link generation during the query execution in order to allow distributed joins.

Our distribution framework will rely on query rewriting. This way, we do not need to index external information locally and avoid data synchronization problems, and we can apply vocabulary mappings to match the remote datasets' vocabularies to overcome vocabulary interoperability issues. Thus, we are able to tackle two problems at once. The main idea behind our distribution framework is to rewrite incoming queries in such a way that the symmetry and transitivity of *owl:sameAs* is exploited. To achieve this, our framework traverses the *owl:sameAs* links of the resources using SPARQL. Incoming queries are split in sub-queries, which can be evaluated independently. For each sub-query, we look for possible *owl:sameAs* linked resources. These sub-queries are refactored to also target the remote SPARQL endpoints of these *owl:sameAs* linked resources. At the same time, we apply vocabulary mappings to the refactored sub-queries to match the remote datasets' vocabularies. Thus, to distribute the queries, some prior knowledge is needed:

- SPARQL endpoints: for each discovered related resource, we have to know the SPARQL endpoint we can consult for retrieving information on the related resource.
- Vocabulary mappings: we cannot just distribute the same sub-query to the different SPARQL endpoints, because every endpoint uses its own vocabulary to describe things, thus we need appropriate mappings for disparate classes and properties of the remote data sources. Some datasets use, e.g., rdfs:label for denoting the name of a person, other datasets use, e.g., foaf:name.

This prior knowledge can be retrieved from the provenance of the *owl:sameAs* links. The provenance of these *owl:sameAs* links can be expressed as rules in the form of SPARQL construct queries, which will be explained in detail in Section 5.4. This allows us to build a lookup table, called *distribution index* which stores the service endpoints and for

each service endpoint the property mappings and the class mappings. This information can be extracted from the SPARQL construct queries, representing the *owl:sameAs* links' provenance. This index building process takes place prior to the queries and not during the queries. It is a pre-processing step. This lookup table is part of the distributed SPARQL processor, which will use this information to answer its incoming queries and to distribute the sub-queries accordingly.

Our distribution framework is implemented as an extension to ARQ. It has two main building blocks. The *index builder* processes the *owl:sameAs* links' provenance to build up the *distribution index*. This is explained in Section 5.4. The *query distributor* is the second main building block, responsible for federating and mapping the incoming queries, using the *distribution index*. This block is explained in Section 5.5. This is schematically shown in Section 5.3.

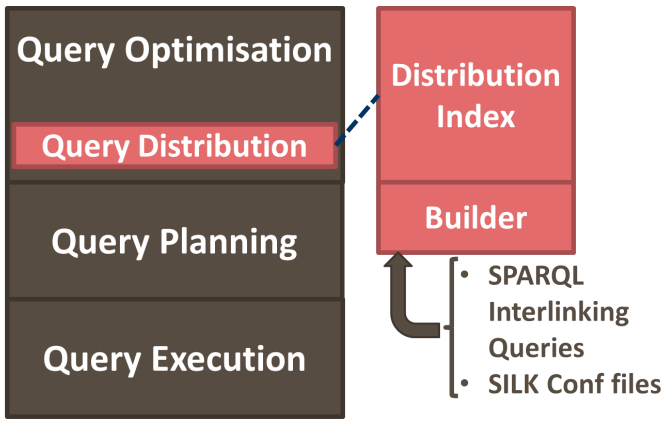

Figure 5.1: Schematic Overview of the Federation Architecture

5.4 Index Builder

The *Index Builder* is responsible for building the *distribution index*. This index must be built before any queries are being fired. The *index builder* will take provenance information of the *owl:sameAs* links to extract the services to which it will distribute incoming queries and for each service it will extract possible mappings.

Many *owl:sameAs* links are generated based on some rule, representing the provenance of a *owl:sameAs* link. RDF Fragment 5.1 shows how a *owl:sameAs* link, relating two persons, can be represented as a SPARQL CONSTRUCT query. The *index builder* actually processes such rules to feed the *distribution index*. Our framework implements two instances of this *index builder* as an extension to ARQ. There are, thus, two ways of feeding the *distribution index*:

- A *SPARQL index builder* that is fed with SPARQL construct queries, representing the provenance of the *owl:sameAs* links present in the dataset. In fact, these rules (SPARQL CONSTRUCT queries) are the queries used for interlinking the dataset. The SPARQL queries are compiled into SPARQL algebra and the *index builder algorithm*, described below, is implemented such that it operates on SPARQL algebraic expressions.
- A *SILK index builder* that takes as input a configuration file of SILK [103], the link discovery framework[6]. This configuration file has all the information available to represent the provenance of the discovered links as SPARQL construct queries. Hence, an *index builder* was also implemented for this sort of input. This *SILK index builder* just parses the SILK configuration file and directly fills up the index, because the SPARQL endpoint, class mappings and property mappings for this SPARQL endpoint are directly available from this configuration file.

RDF Fragment 5.1: Example owl:sameAs Provenance

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX lons: <http://foo.org/localontologynamespace#>
CONSTRUCT {?resource owl:sameAs ?remoteresource}
WHERE {
  ?resource   a              owl:Thing.
  ?resource   lons:type      "person".
  ?resource   lons:name      ?concept.
  ?concept    skos:prefLabel ?name.
  SERVICE <http://dbpedia.org/sparql>
  { ?remoteresource     a              foaf:Person.
    ?remoteresource     foaf:name      ?remotename.
    FILTER (str(?remotename) = ?name } }
```

[6] http://www4.wiwiss.fu-berlin.de/bizer/silk/

5.4.1 Index Builder Algorithm

As explained, the provenance information of all the *owl:sameAs* links of a dataset can be expressed as a number of SPARQL queries. The *index builder* iterates through all these queries and for each query it does the following:

1. Extract the remote SPARQL endpoint from the SERVICE element and store it in the *distribution index*. In our example this is <http://dbpedia.org/sparql>.
2. Extract the query variables from the CONSTRUCT clause, expressing the local resource (e.g., ?resource) and remote resource (e.g., ?remoteresource).
3. Extract the triple pattern for the local resource and the triple pattern for the remote resource.

```
Local triple pattern: ?resource a owl:Thing.
              ?resource lons:type "person".
              ?resource lons:name ?concept.
              ?concept skos:prefLabel ?name

Remote triple pattern: ?remoteresource a foaf:Person.
              ?remoteresource foaf:name ?remotename.
```

4. Extract possible FILTER expressions for the local resource and the possible FILTER expressions for the remote resource.

```
FILTER (str(?remotename) = ?name)
```

5. Extract all different paths from both the local and remote triple patterns. These paths can include FILTER expressions, thus for building the paths also the extracted FILTER expressions are taken into account.

```
Local path 1: ?resource a       owl:Thing.
Local path 2: ?resource lons:type "person".
Local path 3: ?resource lons:name ?concept.
              ?concept skos:prefLabel ?name

Remote path 1: ?remoteresource a foaf:Person.
Remote path 2: ?remoteresource foaf:name ?remotename.
               FILTER (str(?remotename) = ?name)
```

6. For every path, extracted from the remote triple pattern and FILTER expressions, that ends in a query variable, find the corresponding path, extracted from the local triple pattern and FILTER expressions, that ends with the same query variable. The part of the remote path, starting from the remote query variable to the query variable, and the part of the local path, starting from the local query variable, are property mappings of each other, as shown in the example below. This mapping is stored in the *distribution index* for this SPARQL endpoint.

```
Property mapping 1:
?resource lons:name ?concept.
?concept skos:prefLabel ?name
=
?remoteresource foaf:name ?remotename.
FILTER (str(?remotename) = ?name)
```

7. The remaining paths from the remote and local triple patterns and FILTER expressions, make up the class mapping. This is also stored in the *distribution index* for this SPARQL endpoint.

```
Class mapping 1:
?resource       a       owl:Thing.
?resource lons:type "person"
=
?remoteresource a foaf:Person.
```

To summarise, we list here the information to be stored in the *distribution index* using our *index builder* algorithm:

1. SPARQL endpoint: http://dbpedia.org/sparql
2. Class mapping: ?resource lons:type "person". ?resource lons:name ?concept. = ?remoteresource a foaf:Person.
3. Property mapping: ?resource lons:name ?concept. ?concept skos:prefLabel ?name = ?remoteresource foaf:name ?remotename. FILTER (str(?remotename) = ?name)

5.5 Query Distributor

The *query distributor* refactors the incoming query to become a distributed query using the information of the *distribution index* built by our *index builder*. The query will be transformed in such a way that

all linked resources via *owl:sameAs* become interchangeable, which aggregates all the information that directly or indirectly is available on these resources. Thus, our framework actually makes use of the fact that *owl:sameAs* is symmetrical and transitive.

This distribution of the incoming query is done during the query optimisation in ARQ, thus after the incoming query has been compiled to a SPARQL algebraic expression. The *query distributor* is implemented as a query transformation that uses the *distribution index* to perform its transformations. It will search for triple patterns that are not part of a SPARQL SERVICE element. Triple patterns belonging to a SPARQL SERVICE operator are not altered, only the part of the incoming query that is meant to be evaluated against the local data is affected by the *query distributor*. The *query distribution algorithm* will distribute incoming queries in two phases. First, the basic graph patterns (BGPs) of the query are being distributed during the *Transform BGP phase*. Later on, these distributed BGPs will be merged appropriately during the *Merge BGP phase*. The details of the *query distribution algorithm* are discussed hereafter.

RDF Fragment 5.2: Example Query Distribution

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbpediaont: <http://dbpedia.org/ontology/>
SELECT DISTINCT ?author ?booktitle ?producer ?comment
WHERE {
  {
   ?book  dbpediaont:writer ?author.     | BGP 1
   ?book  rdfs:label ?booktitle.         |
   ?book  rdfs:comment ?comment.         |
  } OPTIONAL
  {
   ?book  dbpediaont:producer ?producer. | BGP 2
  }
}
```

5.5.1 Transform BGPs Algorithm

This algorithm operates on a SPARQL algebraic expression and will, as explained, transform first the BGPs individually. This distribution is illustrated with an example. In this example, the index is built from the query rule, shown in RDF Fragment 5.1, which was discussed in detail

in Section 5.4. The incoming query, this algorithm is illustrated with, is shown in RDF Fragment 5.2. The *query distribution algorithm* does for every BGP of the incoming query the following:

1. Extract the triples of the BGP to be evaluated to the local data (i.e., triples from triple patterns not part of a SPARQL SERVICE element).

```
BGP 1 triple pattern: ?book  dbpediaont:writer ?author.
              ?book  rdfs:label ?booktitle.
              ?book  rdfs:comment ?comment.

BGP 2 triple pattern: ?book  dbpediaont:producer ?producer
    .
```

2. Extract possible FILTER expressions affecting these extracted triple patterns.
3. Extract all nodes from the BGP. These nodes can be URIs or query variables (e.g., ?book, ?booktitle, ?author, and ?comment for BGP 1 from our example depicted in RDF Fragment 5.2).

```
BGP 1 nodes: ?book, ?author, ?booktitle, ?comment.

BGP 2 nodes: ?book, producer.
```

4. For every node, extract all possible paths (i.e., *query paths*) from the extracted triples and FILTER expressions. Thus, these query paths can include FILTER expressions, apart from the triples. They can be seen as property paths, extended with FILTER expressions.

```
Query paths extracted with starting node ?book from BGP 1:
Query path 1: ?book dbpediaont:writer ?author.
Query path 2: ?book rdfs:label ?booktitle.
Query path 3: ?book rdfs:comment ?comment.

Query paths extracted with starting node ?book from BGP 2:
Query path 1: ?book  dbpediaont:producer ?producer.
```

5. Every extracted path from the previous step has a local variant (e.g., ?book dbpediaont:writer ?author) and a remote variant (e.g. ?book owl:sameAs ?DQV0. SERVICE <http://dbpedia.org/sparql> ?DQV0 dbpediaont:writer ?author), which are tied together using the UNION operator, as

RDF Fragment 5.3: Part of the Outcome from step 5 of the Transform BGPs Algorithm

```
Distributed query paths extracted with starting node ?book from
     BGP 1:
(sequence
 (union
  (bgp (triple ?book dbpediaont:writer ?author))    |Local QP1
  (sequence
   (bgp (triple ?book owl:sameAs ?DQV0))            |Remote QP1
   (service <http://dbpedia.org/sparql>             |
    (bgp (triple ?DQV0 dbpediaont:writer ?author))))|
 (union
  (bgp (triple ?book rdfs:label ?booktitle))        |Local QP2
  (sequence
   (bgp (triple ?book owl:sameAs ?DQV1))            |Remote QP2
   (service <http://dbpedia.org/sparql>             |
    (bgp (triple ?DQV1 rdfs:label ?booktitle))))    |
 (union
  (bgp (triple ?book rdfs:comment ?comment))        |Local QP3
  (sequence
   (bgp (triple ?book owl:sameAs ?DQV2))            |Remote QP3
   (service <http://dbpedia.org/sparql>             |
    (bgp (triple ?DQV2 rdfs:comment ?comment))))))  |
```

shown in RDF Fragment 5.3. For the remote variant of the query path, the start node of that query path is decoupled into the *owl:sameAs* equivalent. The *owl:sameAs* equivalent is distributed to the SPARQL endpoints from the *distribution index* and at the same time mappings from the *distribution index* for that SPARQL endpoint are applied. The example depicted in RDF Fragment 5.3 shows the query paths extracted from node ?book for the example query depicted in RDF Fragment 5.2.

6. The outcome of the previous step are only the distributed query paths starting from a certain node (e.g., ?book in our example). These blocks (i.e., sequence operators) still need to be tied together. This is done during this last step using the UNION operator. This way, a BGP is split up in all possible query paths. Figure 5.2 gives a schematic overview of how a BGP 1 from the example depicted in RDF Fragment 5.2 is distributed. Each of the four right blocks represent the outcome from the previous step, but each using a different starting node.

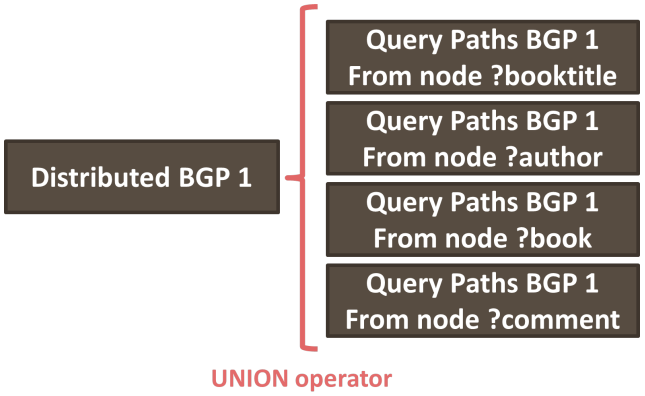

Figure 5.2: Schematic Overview of the Distributed BGP 1

5.5.2 Merge BGP Algorithm

The previous phase only distributed the BGPs. These distributed BGPs cannot be merged as such. If we take back the example depicted in RDF Fragment 5.2 and consider BGP 2, the nodes extracted from this BGP are only ?book and ?producer. If the local dataset only has information on authors (which are linked to their DBpedia *owl:sameAs* equivalent) and we want to have the producer of a book, there needs to be a query path from ?author to ?producer. This query path is not extracted from BGP 1, nor from BGP 2. For this, we need a special merging algorithm. The result of such a merge is shown in Figure 5.3. This example only shows the merged operator with query path starting with node ?book. The result is an extra query path, merged using a LeftJOIN operator. To achieve this, our merging algorithm works as follows:

1. The triples from BGP 1 and BGP 2 are merged.
2. From the merged triples all nodes are extracted (e.g. ?book, ?booktitle, ?comment, ?author and ?producer).
3. For every node, all query paths are extracted.
4. If a query path from a certain node is not yet present in the distributed BGP 1, then the query path is merged with the already extracted query paths for that node in the distributed BGP 1. This merge is done using the operator that binds BGP 1 and BGP 2, i.e., LeftJOIN operator in this case, which is the algebraic equivalent of an OPTIONAL operator in SPARQL, as shown in Figure 5.3.

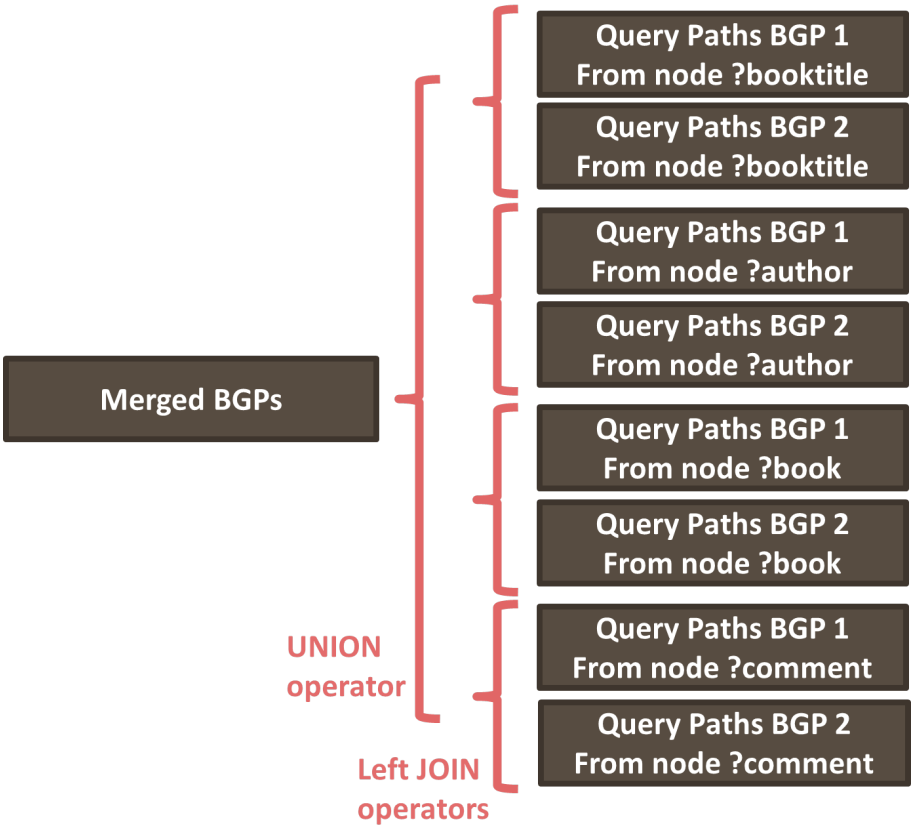

Figure 5.3: Schematic overview of the merged BGPs

RDF Fragment 5.4: Merging of Distributed BGP 1 and BGP 2 - only that part with query paths starting with ?book node

```
Merged (BGP 1 and BGP 2), distributed query paths extracted
    with starting node ?book:
(leftjoin
(sequence operator shown in step 5 of the transform BGP
    algorithm )
 (union
    (bgp (triple ?book dbpediaont:producer ?producer))
    (sequence
      (bgp (triple ?book owl:sameAs ?DQV0))
      (service <http://dbpedia.org/sparql>
        (bgp (triple ?DQV0 dbpediaont:producer ?producer))))))
```

RDF Fragment 5.5: Merging of distributed BGP 1 and BGP 2 - only that part with query paths starting with ?author node

```
Merged (BGP 1 and BGP 2), distributed query paths extracted
    with starting node ?outhor:
(leftjoin
(sequence operator for the node ?outhor )
 (union
    (bgp (triple ?book dbpediaont:writer ?author)
        (triple ?book dbpediaont:producer ?producer))
    (sequence
      (bgp (triple ?author owl:sameAs ?DQV0))
      (service <http://dbpedia.org/sparql>
        (bgp (triple ?book dbpediaont:writer ?DQV0)
             (triple ?book dbpediaont:producer ?producer))))))
```

5.6 Optimisations

So far, the basic federation algorithm has been discussed. It is aimed at distributing the incoming query to the fullest, in order to find an answer. A feature of this query distribution framework is that it does not narrow down its search space. Thus, if an answer is out there, it will be returned. This feature comes at the cost of query execution performance. Optimisations should be focused on increasing the query execution performance, without narrowing down the search space, in contrary to many other query federation frameworks which optimise by narrowing down the search space, e.g., FedX. Practically, this means limiting the SERVICE operations when distributing the query paths and increasing their execution. First we will discuss two optimisations that are already in place. Next we discuss some future optimisations.

A. Query Paths

The basic entity our distribution algorithm operates on is a query path. A query path is actually a property path extended with FILTER expressions and GRAPH expressions. These query paths are distributed to the remote SPARQL endpoints and, hence, FILTER and GRAPH expressions are evaluated at the remote SPARQL endpoint. At the same time these query paths, which are fired repeatedly, better exploit the Web caching of the SPARQL queries.

B. Mappings

To the query path that is being federated, mappings are applied to refactor the query path to the vocabularies the remote dataset supports. This enhances interoperability of the query federation framework and makes our framework more robust. The mappings are retrieved implicitly, by processing the provenance of the *owl:sameAs* links. As a consequence, the mappings are always expressed in terms of the local vocabularies used. It can be seen either as a point-to-point mapping strategy or as a strategy using a universal data model, where the universal data model is always the local data model from which the distributions start.

C. Minimising SERVICE Operations, optimising the query execution

Minimising the number of SERVICE operations is a future goal of our framework. This means we need to filter out the unnecessary query paths, which do not contribute to an answer. First of all, we could group the queries for the same endpoint to already minimise the number of service calls. Another optimisation will be the introduction of VoID descriptions to filter out the unnecessary service calls. Considering the optimisation of the query execution, one could look at the VoID descriptions to optimise the query planning of the incoming queries. Another contribution to the query planning could be the analysis of the remote SPARQL endpoints. Some COUNT queries on the properties and classes that are represented in the *distribution index*, can optimise the query planning and, hence, the query execution. Another possible optimisation is a concurrent execution of the query processing, especially the service operations. Until now, SPARQL query processing in ARQ is synchronous, but non-blocking. This means that the result of a query is already available during query execution and communication with the client can thus be made concurrent, but this is only effective if the processing of the federated query also happens concurrently. By introducing a SPARQL SERVICE operator, which is processed concurrently, the SPARQL processing becomes concurrently. By doing this, results become available as soon as the SPARQL processor has found some answers, but there are also some benefits regarding the performance of the SPARQL processing. Looking at the schematic overview of a distributed query, shown in Figure 5.2, a query is split up in different versions of the query from the perspective of a node that needs to be decoupled into its *owl:sameAs* equivalents (UNION

operator in Figure 5.2). All these different query versions, of which two are depicted in RDF Fragment 5.4 and RDF Fragment 5.5, need to be joined. This join can be replaced by our concurrent operator, which will evaluate all the different versions of the query concurrently. Our operator, is in fact a thread-pooled UNION operator, merging all its incoming results, and the SERVICE operator.

5.7 Evaluation

For evaluating our distribution framework, we make use of the Berlin SPARQL Benchmark. The dataset of this benchmark, consisting of 10.000 product descriptions, is published over two SPARQL endpoints, i.e., a local and a remote SPARQL endpoint. We have set up a local SPARQL endpoint, whose dataset only contains product URIs, linked via *owl:sameAs* with the product resources of the remote BSBM dataset. The BSBM query-mix is fired at the local SPARQL endpoint, which distributes the queries using our distribution framework to solve the queries. For this, it contacts the remote SPARQL endpoint. The execution times of the results are used for evaluation. These result times are evaluated against the result times of the 'perfect' distributed query, see Table 5.1. This 'perfect' distributed query just forwards the whole query of the BSBM query-mix to the remote SPARQL endpoint using the SERVICE operator. This way, we evaluate the result times of a query once using our distribution algorithm against the result times of a query using a 'perfect' query distribution. By doing this kind of evaluation, our evaluation is also independent of the used hardware/software and the size of the BSBM dataset and thus evaluates the distribution algorithm itself. Table 5.1 shows the results for only a part of the queries of the BSBM query-mix. The other query results showed similar performance. These results were obtained by running the queries 100 times and calculating their average response times. We did not opt to run the queries 10.000 times as is usual in these cases, because then our results would be influenced too much by the web caching, and we want to evaluate our algorithm. The results for query 7 show a large deviation, which is due to the fact that a large number of intermediate results were transmitted during its evaluation. This shows our algorithm has still some room for optimisation.

'perfect' distribution algorithm	Result time first result	Result time last result
Query 1	780 ms	3763 ms
Query 3	2701 ms	4076 ms
Query 7	517 ms	603 ms
our distribution algorithm	Result time first result	Result time last result
Query 1	967 ms (+23%)	4480 ms (+19%)
Query 3	3594 ms (+22%)	4929 ms (+21%)
Query 7	631 ms (+22%)	862 ms (+42%)

Table 5.1: Performance Results of our Query Federation Framework

5.8 Conclusion

In this chapter, we described our framework for distributing queries relying on the symmetry, transitivity, and the provenance information of *owl:sameAs*. This way, our framework acts like a window on the Linked Open Data cloud, where all information is available on resources that are directly (via *owl:sameAs* links) linked to one of your resources, or indirectly (via a *owl:sameAs* links of a linked resource). For this, the framework relies on the provenance of the *owl:sameAs* links, which can be expressed by a rule or SPARQL construct query. The rules or queries give information on the SPARQL endpoint to use for querying that linked dataset and some vocabulary mappings to use for querying that dataset.

Our framework is implemented as an extension of ARQ and consists of two main components: the *index builder*, and the *query distributor*. The *index builder* will build an index based on the provenance of the *owl:sameAs* links. It supports building an index based on SPARQL construct queries or rules, which express the provenance of the generated *owl:sameAs* links, and it supports also building an index based on a configuration file for the SILK enrichment framework. The *query distributor* will re-factor the incoming queries to distributed sub-queries, targeting the SPARQL endpoints of the linked datasets, during the query optimisation phase. During this re-factoring the appropriate mappings are applied on the triple patterns, mapping the triple patterns to vocabularies the remote dataset supports. This approach gives data providers more control. They can easily set-up their own distributed SPARQL endpoint, which will distribute incoming queries to

those datasets they trust, because these are the datasets they use for interlinking. We evaluated our platform using the BSBM benchmark, but in a distributed environment. For this the queries from the benchmark had to be distributed to answer the queries.

The author's work on query federation led to the following publications:

- Sam Coppens, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. „Querying the Linked Data Graph using owl: sameAs Provenance". In: *Proceedings of the 16th International Conference on Model Driven Engineering Languages and Systems* (2013)
- Sam Coppens, Miel Vander Sande, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. „Remote Reasoning over SPARQL". in: *Proceedings of the 6th Workshop on Linked Data on the Web* (2013)

Chapter 6

Self-Sustaining Platforms - A Distributed Reasoning Framework to implement a Semantic Workflow Engine

In this chapter, we provide a novel semantic workflow system, based on semantic functional service descriptions and a rule file. The idea for this workflow engine came from the long-term preservation archive, which could benefit greatly from such a semantic workflow engine. The workflow engine follows a three-step process. First, it determines for all the resources in its knowledge base the functionality they need to progress in the workflow. This uses a phase-functionality rule file which binds phases of the workflow to functionalities. During a second phase, the functionalities are mapped to REST service calls using RESTdesc's functional descriptions. During the third step, the engine executes the generated service calls and pushes the resource it acted on to the next phase in the workflow using a phase-transition rule file. The main advantage of this approach is that each step can be influenced by external information from the Linked Open Data cloud. It exploits the fact that Linked Open Data and RESTful Web services and APIs are all resource-oriented. Moreover, the workflow rule file makes the system easily adaptable and extensible to achieve new functionalities or to obey changing company policies. Finally, the separation between functional descriptions and service descriptions supports easy management over the fast-changing services.

6.1 Introduction

Today, applications on the Web increasingly rely on Linked Open Data [100] and RESTful services [110]. Both have a resource-oriented architecture that exploit links between resources. The increasing speed at which Linked Open Data, Web services and APIs are being deployed, demands an intelligent, expandable workflow engine that can be used in various domains, such as *factories* and *Smart Cities*. Process control in factories is often hard-coded into the control software of the production machines. Whenever they want to introduce a new sensor to steer the process, software needs to be adapted, recompiled, and re-deployed on every machine. An even bigger problem arises for Smart Cities, which use millions of wireless sensors. Existing workflow engines try to make this manageable, yet they are mostly manually created. Moreover, the intelligence of a platform is often distributed over the software code and the workflow engine, making the management of such a system a huge burden. Novel workflow engines should be able to integrate resources of any kind, i.e., Linked Open Data and RESTful Web services, to even become resource-agnostic. The only difference is that resources describing RESTful Web services and APIs get a functionality attached to them.

In the solution we envision, the software intelligence and the workflow engine are *unified* into one system. Its first cornerstone is that workflows do not require manual composition anymore, since they are composed automatically using *semantic service descriptions* and *workflow description files* that captures the intelligence of the platform. By relying on functional service descriptions, services offering similar functionality become interchangeable. The declarative workflow rule files steer the reasoning engine by describing the whole process in terms of functionalities. This way, services become interchangeable and the platform becomes a loosely coupled system. Adaptations in the services or the workflow rule files are straightforward, without the need to adapt software code.

The second cornerstone of the envisioned platform is the *automated execution* of generated workflows. Together with the automatic workflow composition, the whole system becomes self-sustaining, easily adaptable and extensible, and highly portable. Workflows are dynamically recomposed when needed and automatically triggered when needed.

This unique combination enables our system to form the backend platform for a broad spectrum of applications, ranging from home domotic systems, long-term preservation platforms, process control software for factories to large Smart Cities. The only assets that need to be managed are the service descriptions and the worfklow description rule files. At any moment new services can be described, which will automatically be incorporated into the newly generated workflows. Even the whole process can be changed, by making alterations to the workflow description rule files. All this happens in a declarative way, eliminating the need to redesign, recompile, and redeploy the software, keeping the system more manageable. These two characteristics, i.e., manageability of the services and speed-up of the development process or extensibility of the system, are key to our solution and distinguish it from traditional workflow engines.

The Chapter is structured as follows: in the next section, we give an overview of existing related work. Section 6.3 will elaborate on our solution and architecture. Next, we discuss all operational phases of the workflow engine. In Section 6.4, we detail the *Functionality Generation Phase*. In Section 6.5, the *Service Generation Phase* is discussed in detail. Section 6.6 describes the *Service Execution and Transition Phase* of our workflow engine. Finally, we discuss some advanced features of the workflow engine in Section 6.7, and we end the Chapter with a conclusion.

6.2 Related Work

Our self-sustaining platform relies on two basic technologies: semantic service descriptions and automatic workflow composition. "Semantic Web Services: a RESTful approach" [111] has a RESTful grounding ontology that maps OWL-S [112] and WADL [113]. OWL-S is a W3C submission for Semantic Web service descriptions. Web Application Description Language (WADL) is a W3C submission for describing REST over HTTP Web services. WADL is to REST what WSDL is to SOAP.

Next to these, the Linked Data API is also a related technology. Their scope is to provide simple RESTful APIs over RDF graphs to bridge the gap between Linked Data and SPARQL. This specification aims to fill that gap by defining an easy to use and easy to deploy API layer that can act as a proxy for any SPARQL endpoint.

When it comes to automatic workflow composition, Data-Fu, a language and interpreter for interaction with read/write Linked Data, is related to our research. Data-Fu is a declarative rule language for specifying interactions between web resources. Another platform, supporting automatic workflow composition is the work of Krummenacher [114]. In this work, he investigates the composition of RESTful resources in a process space. The work is based on resources described by graph patterns. Similar work is carried out by Speiser and Harth [115], which also relies on graph patterns for RESTful Linked Data Services. For our workflow composition, we rely on rule-based reasoning. All the intelligence of an application is gathered by three rule files. This centralisation of the application's intelligence allows easy management of the services, because services become interchangeable if they provide the same functionality. This is already a major contribution, comparing to the other three automatic workflow composition frameworks. At the same time, it allows easy adaptation and extensibility of the application through the adaptation of the three rule files. There is no need anymore to recompile, and redeploy the software. Because of the three-phase operation of our semantic workflow engine, this engine can easily work completely distributed to meet scalability issues.

6.3 Concept and Architecture

When composing a workflow, our platform basically needs the following information, as shown in Figure 6.1:

Knowledge Base The knowledge base contains actually the information to be acted on. This is Linked Open Data, coming from a local triple store or optionally the Linked Open Data cloud itself.

Phase-Functionality rule file A phase-functionality rule file is a declarative file, stating how the different phases of a workflow are mapped to functionalities to move the resource to the next phase in the workflow. Rules are described in N3 [16], as will be discussed in Section 6.4.

Phase-Transition rule file This file is actually responsible for routing the different phases (read functionalities) of a workflow such that it complies to a certain overall functionality or company policy.

Functional service descriptions These service descriptions describe the individual services the workflow engine can make use of to compose its workflows. RESTdesc [116] describes services, as will be detailed in Section 6.5.

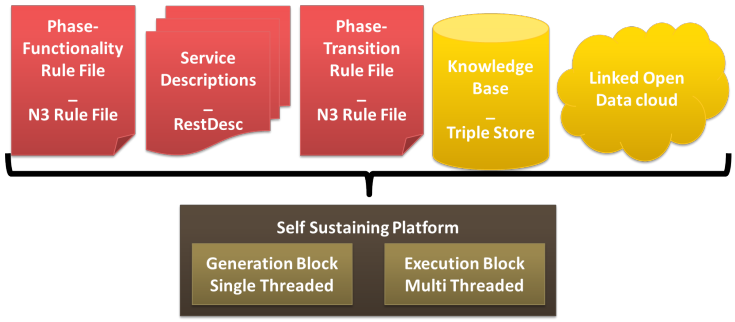

Figure 6.1: Basic building blocks of the self-sustaining platform.

The phase-functionality rule file, the phase-transition rule file, and the service descriptions in combination with the knowledge base, and optionally even external knowledge bases from the LOD cloud will generate the needed service calls by means of reasoning. This happens by the workflow generator. Because we rely on monotonic rule reasoning, this is a single threaded process. The generated service calls will be executed by the workflow executor. Of course, this can be a multi-threaded process.

The operation behind the self-sustaining platform actually consists of three phases, as illustrated in Figure 6.2:

1. In the **functionality generation phase**, the phase-functionality rule file will tell for each resource which functionality it needs to get to the following phase of the workflow. This will happen by reasoning with the rule file over the knowledge base. This will entail triples describing a functionality the resource needs to get to the following phase of the workflow.
2. In the **service call generation phase**, these functionalities are mapped to service calls using the service descriptions. This is also done by reasoning over the entailed triples of the previous phase. The entailed triples of this reasoning cycle will describe service calls to be executed during the following phase. Splitting up the functionalities of service calls has a great advantage for the management of the services and generated workflows, because now workflow management and service management are split up. This split up decouples workflows from the services used in them, which is great benefit, certainly when hosting millions of services as service providers do.

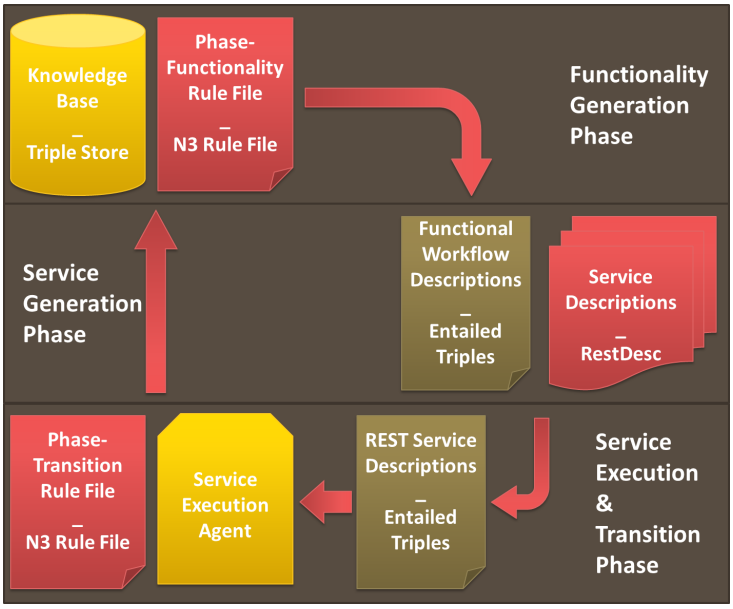

Figure 6.2: Schematic operation of the self-sustaining platform.

3. In the **service call execution phase**, the generated service call is being executed. If the service executed successfully, the phase-transition rule file is used to move to the next phase within the workflow.

Determining the needed transition is performed by means of reasoning over the entailed triples from the previous phase. The entailed triples of this phase will be stored to the knowledge base and a whole new reasoning cycle from the first phase is started. The next sections, will discuss the different steps in detail.

To demonstrate our idea, we will focus on the use case of publishing Linked Open Data. Publishing Linked Open Data typically consists of four steps: harvesting the data, mapping the harvested data, reconciling the mapped data, enriching the reconciled data, and finally, publishing the enriched data.

Figure 6.3: Example workflow for publishing LOD.

6.4 Functionality Generation Phase

This phase-functionality rule file will split up the workflow into different phases and it will list the functionality it needs for each phase to get to the following phase. If we take back the example of publishing Linked Open Data, then its workflow consists of the following ordered list of functionalities: harvesting, mapping, reconciliation and publication. Each of the functionalities give rise to a new phase, as schematically shown in Figure 6.3. The nodes denote the phase, the edges the functionality it needs to transition to a following phase.

The phases, and the functionalities connecting them, form a functional workflow description. This is a composition of functionalities that need to be performed in order to fulfil a certain policy or achieve a certain goal. Such a goal or policy can be described using a rule file and a rule-based reasoner. In this rule file, the phases and functionalities need to be denoted with URIs. Taking back our example of our LOD publisher, the rule file looks like the following:

```
@prefix wf: <http://example.org/workflow#>.
@prefix fn: <http://example.org/functionalities#>.

{?object0 wf:hasStarted wf:phase_0} =>
                         {?object0 fn:harvestRecord ?object1.
                          ?object1 wf:hasCompleted wf:phase_0}.

{?object1 wf:hasStarted wf:phase_1} =>
                         {?object1 fn:map ?object2.
                          ?object2 wf:hasCompleted wf:phase_1}.

{?object2 wf:hasStarted wf:phase_2} =>
                         {?object2 fn:reconcile ?object3.
                          ?object3 wf:hasCompleted wf:phase_2}.

{?object3 wf:hasStarted wf:phase_3} =>
                         {?object3 fn:enrich ?object4.
                          ?object4 wf:hasCompleted wf:phase_3}.

{?object4 wf:hasStarted wf:phase_4} =>
                         {?object4 fn:publish ?object5.
                          ?object5 wf:hasCompleted wf:phase_4}.
```

This phase-functionality rule file actually describes the different phases of a resource in a workflow and for each phase its functionality. It does not tell how the different phases should be coupled in order to achieve a certain workflow. These transitions are described in the phase-transition rule file, discussed in Section 6.6. During the second step, this functional workflow, actually described as entailed triples, is being materialised to explicit service calls. Assume we have an URL of a resource we want to publish as LOD, e.g., <http://foo.org/resource/1234>. If we add the following triple to our triple store/knowledge base, we get the publishing procedure starting:

```
<http://foo.org/resource/1234> wf:hasStarted wf:phase_0.
```

Reasoning over this triple with our example rule file yields the following triples:

```
<http://foo.org/resource/1234> fn:harvestRecord ?object1.
?object1 wf:hasCompleted wf:phase_0.
```

These triples are passed to the next step, where functionalities are translated into REST service calls by means of reasoning over these entailed triples with the service descriptions. Once, the service calls are entailed, they are passed to the execution phase. After the successful execution, all the entailed triples are added to the knowledge base, which finishes a first reasoning cycle of the platform. After this step, a new reasoning cycle is fired. Another reasoning cycle over these triples would yield the following triples, which in turn are passed to the next step:

```
?object1 fn:map ?mappedResult.
?mappedResult wf:hasCompleted wf:phase_1.
```

Eventually, after several reasoning cycles, the entire workflow will be executed.

6.5 Service Generation Phase

As explained above, this step is going to turn the result of the functionality generation phase into a service call description, which will be fed to the service execution and transition phase. The result of the

functionality generation phase is actually a set of entailed triples, as shown in the previous section. For each phase in the workflow, we have a triple denoting the functionality it needs for the next phase. These triples should actually result in a description of a REST service, which serves the functionality.

For the service description, we rely on RESTdesc [116]. RESTdesc is both a description and a discovery method targeting restful Web services, with an explicit focus on functionality. It consists of well-established technologies such as HTTP and RDF/Notation3 and is built upon the concepts of hyperlinks and Linked Data. Its goal is to complement the Linked Data vision, which focuses on static data, with an extension towards Web services that focus on dynamic data. All RESTdesc descriptions are:

- *self-describing*: using Notation3 semantics;
- *functional*: explaining exactly what the operation does;
- *simple*: descriptions are expressed directly using domain vocabularies.

Since RESTdesc entails the operational semantics of Notation3, it allows for versatile discovery methods. We can indeed use the power of Notation3 reasoners to determine whether a service satisfies at set of conditions. Even more advanced reasoning is possible to decide on service matching, and/or to create complex compositions of different services. We see this as an important prerequisite for services in order for them to contribute to the future Web of Agents, since new functionality can only be obtained by on-demand compositions tailored to a specific problem. Such a RESTdesc description of a service that harvests a record, looks like this:

```
@prefix fn: <http://example.org/functionalities#>.
@prefix http: <http://www.w3.org/2011/http#>.

{ ?url fn:harvestRecord ?record. }
=>
{
  _:request http:methodName "GET";
            http:requestURI ?url;
            http:resp [ http:body ?record ].
}.
```

It actually says: If you have a URL from which you want to harvest a metadata record, then you can do this by an HTTP GET request on the URL and the response will contain the record in its body. POST requests are also described in this way. The following mapping service is a good example of a POST request.

```
{
  ?record fn:map ?mappedRecord.
}
=>
{
  _:request http:methodName "POST";
            http:requestURI "http://example.org/service/map/";
            http:body ?record;
            http:resp [ http:body ?mappedRecord ].
}.
```

Thus, during this second step, we only need to reason with the RESTdesc service descriptions over the entailed triples of the first step in order to get the service call descriptions that need to be executed during the next step. If we take back the example of Section 6.4, the entailed triples from the first reasoning cycle being fed to this step are:

```
<http://foo.org/resource/1234> fn:harvestRecord ?result.
?result wf:hasCompleted wf:phase_0.
```

Reasoning over these triples with the service description of the harvest service, inferences the following triples that are fed to the workflow execution step:

```
_:request http:methodName "GET";
          http:requestURI <http://foo.org/resource/1234>;
          http:resp [ http:body ?record ].
```

If you have multiple services covering the same functionality in your repository with RESTdesc service descriptions , this phase would yield multiple REST service calls being described. Not all these service descriptions need to be fed to the service execution phase, only one needs to. For this reason, this phase stops after the first RESTdesc service description that entails a service call description. This separation of functionalities and services that cover these functionalities greatly enhances the management of the services and the workflows. It makes

services with the same functionality interchangeable. Of course, this service selection step can also be influenced, as will be discussed in Section 6.7.4.

6.6 Service Execution and Transition Phase

In this step, the entailed triples of the previous step are processed. The entailed triples of the previous step actually describe a REST service call, explaining the HTTP GET request can be done on the URL `http://foo.org/resource/1234` and the record's metadata will be returned into the response's body. Thus, an execution agent can process these entailed triples, describing a service call. If the agent executes the service call successfully, the phase transition rule file is used to progress to the next phase of the workflow. This phase-transition file actually routes all the different phases of the workflow. This is done by reasoning over the entailed triples of the previous steps. For our LOD publication example, this phase transition file looks like this:

```
@prefix wf: <http://example.org/workflow#>.
@prefix fn: <http://example.org/functionalities#>.

{?object wf:hasCompleted wf:phase_0} =>
                        {?object0 wf:hasStarted wf:phase_1}.

{?object wf:hasCompleted wf:phase_1} =>
                        {?object0 wf:hasStarted wf:phase_2}.

{?object wf:hasCompleted wf:phase_2} =>
                        {?object0 wf:hasStarted wf:phase_3}.

{?object wf:hasCompleted wf:phase_3} =>
                        {?object0 wf:hasStarted wf:phase_4}.
```

For our example, the entailed triples fed to this step are:

```
<http://foo.org/resource/1234> wf:hasCompleted wf:phase_0.
  _:request http:methodName "GET";
            http:requestURI <http://foo.org/resource/1234>;
            http:resp [ http:body ?record ].
```

The service-executing agent detects the service call description and executes it. If the execution was successful, the reasoning with the phase-transition file is started, yielding the following triples:

```
<http://foo.org/resource/1234> wf:hasStarted wf:phase_1.
```

After this last reasoning cycle, all the entailed triples of the three steps are being ingested into the knowledge base and a new functionality generation phase is started, as shown in Figure 6.2. If the execution of the service call was unsuccessful, the operation breaks up, storing nothing to the knowledge base, because you cannot state the phase has already been finished. Thus, the triples being added to the knowledge base are:

```
<http://foo.org/resource/1234> wf:hasCompleted wf:phase_0.
_:request http:methodName "GET";
            http:requestURI <http://foo.org/resource/1234>;
            http:resp [ http:body ?record ].
<http://foo.org/resource/1234> wf:hasStarted wf:phase_1.
```

6.7 Advanced Features

In this section, we are going to discuss some advanced features of the workflow engine. Until now, only the basic operations have been discussed. By adapting the phase-transition file, and the service descriptions, some more advanced features are possible in this framework. The advanced features discussed in this section are: how to integrate feedback loops, cron jobs, and nested functionalities. The advanced features are not limited to these ones. Other features are, e.g., the possibility to include Linked Open Data into the workflow, such that workflows are adapted or triggered by external data from the Linked Open Data cloud. SPARQL endpoint results can also be integrated into the workflows. For this, SPARQL requests are modeled as RESTful service calls. The results from these endpoints can then be used within the rule files making the workflows adapt to this external data. Another feature that can be integrated into the workflow engine is basic workload balancing. For this, we can include counters attached to service endpoints. A service selection can then take into account these counters and select the one with the least load. This would already

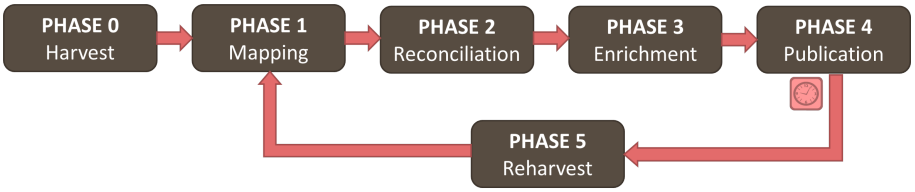

Figure 6.4: Example timed feedback loop for publishing LOD.

fulfil a basic workload balancing. In the remainder of this section, we will discuss in detail the following advanced features: feedback loops with a cronjob (time-based, repeating task), nested functionalities, how to include authority, and how to influence the service selection.

6.7.1 Feedback Loops

Feedback loops can be built into the workflow. In our example, we can introduce an update process by introducing a recurring reharvest of the record, as depicted in Figure 6.4 This is achieved by adapting the first rule and adding a rule to the phase-functionality rule file:

```
@prefix dc: <http://purl.org/dc/elements/1.1/>.
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix time: <http://www.w3.org/2000/10/swap/time#>.

{?object0 wf:hasStarted wf:phase_0} =>
                {?object0 fn:harvestRecord ?object1.
                 ?object1 wf:hasCompleted wf:phase_0.
                 ?object1 dc:source ?object0.
                 ?object1 dcterms:modified time:localTime.}.

{?object5 wf:hasStarted wf:phase_5.} =>
                {?object5 dc:source ?url.
                 ?url fn:reHarvestRecord ?object6.
                 ?object6 wf:hasCompleted wf:phase_5.}.
```

With these rules, we define an extra phase for the workflow, i.e., `wf:phase:5`, which is coupled to the functionality `fn:reHarvestRecord`. Next, this phase needs to be integrated into the phase-transition rule file. The phase-transition file will get the following extra rule:

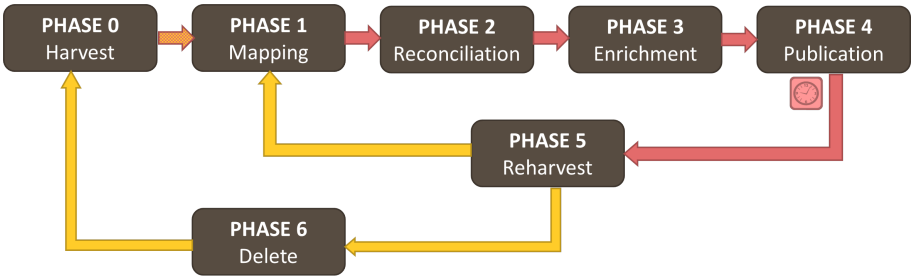

Figure 6.5: Example of a nested loop for publishing LOD.

```
@prefix func: <http://www.w3.org/2007/rif-builtin-function#>
@prefix pred: <http://www.w3.org/2007/rif-builtin-predicate#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

{?object wf:hasCompleted wf:phase_4;
         dcterms:modified ?lastHarvest.
 (time:localTime ?lastHarvest) func:subtract-dateTimes ?
    difference.
 (?difference "P14D"^^xsd:dayTimeDuration)
   pred:dayTimeDuration-greater-than true.} =>
                {?object wf:hasStarted wf:phase_5}.

{?object wf:hasCompleted wf:phase_4} =>
                {?object wf:hasStarted wf:phase_1}.
```

6.7.2 Nesting

The functionalities can be nested using multiple rules, as shown in Figure 6.5. In our previous example, the reharvest functionality consists actually of a new delete and an already existing harvest functionality. The following rules for the phase-functionality rule file shows reharvesting:

```
{?object fn:reHarvestRecord ?result} =>
                   {?object wf:hasStarted wf:phase_6.}.

{?object wf:hasStarted wf:phase_6} =>
                   {?object fn:delete ?delete.
                    ?object wf:hasCompleted wf:phase_6}
```

And the following rules are added to the phase-transition file:

```
{?object wf:hasCompleted wf:phase_6} =>
                    {?object wf:hasStarted wf:phase_0.
                     ?object wf:isReharvest true.}

{?object wf:hasCompleted wf:phase_0} =>
                    {?object wf:hasStarted wf:phase_1.}

{?object wf:isReharvest true.
 ?object wf:hasCompleted wf:phase_0} =>
                    {?object wf:hasCompleted wf:phase_5.}
```

This needs a little explanation. There is no service providing the reharvest functionality, there are services for the delete and harvest functionality. Whenever our semantic workflow engine cannot trigger a rule with the entailed triples from the functionality generation phase during the service generation phase, these entailed triples are taken to the service execution and transition phase. This reasoning cycle, only defines phase transitions without delivering a service call description. During the subsequent reasoning cycle, our newly defined rules will bring our object from the deadlocked reharvest state to the delete state for which it has a service description. In turn, the delete phase will eventually give rise to the harvest phase and this is how functionalities can be nested in our platform. After this harvest, the nested loop needs to be closed. This is done with the last rule from the phase-transition file, which says that if the harvest is finished and is part of a reharvest operation, the reharvest operation is finished.

6.7.3 Authority

It also becomes easy to integrate authorisation into the workflows. To integrate this in the workflows, a new phase must be integrated into the workflow, e.g., in our example, for validating the generated enrichments. The transition from this phase (i.e., its completion) is then actually steered by an external REST service, acting as the authority and validating the enrichments manually. In our example we can decide to publish the record after the reconciliation and do the enrichment in a separate track. Once these enrichments are validated, they are also published.

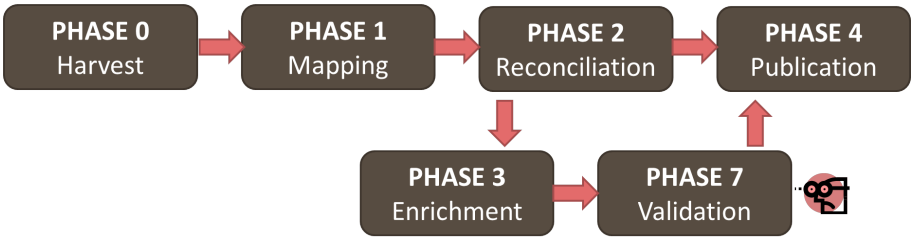

Figure 6.6: Example of a authorised validation step in a workflow.

First of all, we need to introduce a new phase for the validation of the enrichments. This new phase will actually be coupled to functionality, which will validate the generated enrichments:

```
{?object wf:hasStarted wf:phase_7} =>
                    {?object fn:validateEnrichments ?result.
                     ?result wf:hasCompleted wf:phase_7}.
```

After introducing this new phase and functionality, considering we have a service fulfilling the functionality, we need some adaptation to the transitions. Thus, after the reconciliation, the record is published immediately, and the record is sent to generate enrichments and to validate them before publication of the validated enrichments, as depicted in Figure 6.6 This leads to the following adaptations of the phase-transition file:

```
{?object wf:hasCompleted wf:phase_2} =>
                        {?object0 wf:hasStarted wf:phase_4.
                         ?object0 wf:started wf:phase_3}.

{?object wf:hasCompleted wf:phase_3} =>
                        {?object0 wf:hasStarted wf:phase_7}.

{?object wf:hasCompleted wf:phase_7} =>
                        {?object0 wf:hasStarted wf:phase_4}.
```

6.7.4 Service Selection

In Section 6.5, we described what happens if several service descriptions fulfil the same needed functionality. Via the workflow rule file this service generation phase can be influenced to obey certain rules for selecting the appropriate service. A use case for this would the selection of the right mapping service. One can imagine having several mapping services in place, each covering other mapping formats as input and output. Then one can have the following rules in the workflow rule file to select the appropriate mapping service during this service generation phase:

```
{?object wf:hasStarted wf:phase_0} =>
    {?object fn:harvestRecord ?result.
     ?result wf:hasCompleted wf:phase_0.
     ?result ex:vocabulary <http://www.loc.gov/MARC21/slim>}.

{?object wf:hasStarted wf:phase_1} =>
    {?object fn:map ?result.
     ?result wf:hasCompleted wf:phase_1.
     ?result ex:vocabulary <http://purl.org/dc/terms/>}.
```

In this workflow rule file, we specified in the first rule the vocabulary used to describe the record harvested, i.e., MARC XML. In the second rule, we specify that everything needs to be mapped to the vocabulary *dcterms*. In combination with specifying the mapping vocabularies, you can steer the selection of the services used for a functionality too:

```
{ ?record ex:vocabulary <http://www.loc.gov/MARC21/slim>.
  ?record fn:map ?mappedRecord.
  ?mappedRecord ex:vocabulary <http://purl.org/dc/terms/>.}
=>
{ _:request http:methodName "POST";
            http:requestURI "http://example.org/service/map";
            http:body ?record;
            http:resp [ http:body ?mappedRecord ].}.
```

This way service descriptions can be manipulated to also include, e.g., a trust rating or a quality rating. The workflow rule file can then be adapted to always take the most trustworthy service or the most qualitative one.

6.8 Conclusions

In this chapter, we presented a novel workflow engine, based on Linked Data, RESTful Web services and rule-based reasoning. Our platform will generate workflows described in terms of connected functionalities. In a later phase, these functionalities are mapped into service call descriptions, which are finally executed. A main feature of our platform is that it is easily adaptable and expandable. The whole system captures the application intelligence through three N3 rule files: one for describing the different phases of a workflow and per phase the functionality it needs, another one describing the phase transitions, and finally one describing the RESTful Web services the platform relies on. These files can be adapted at run-time, and at the same time aggregate the platform's intelligence.

Another main feature is the explicit separation between functionalities and services. By this, services serving the same functionality become interchangeable. This makes both the management of the workflows and the services a lot easier. Our generated workflows can be steered by external Linked Data, so that the platform can act on this external data. A last feature of the platform is its ability for error handling. These features together make the platform self-sustaining.

The author's work on Self-sustaining platforms and semantic workflows led to the following publications.

- Sam Coppens, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. „Self-Sustaining Platforms - Grid Reasoning". In: *Transactions on Autonomous and Adaptive Systems* (2014)
- Sam Coppens, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. „Self-Sustaining Platforms: a semantic workflow engine". In: *Proceedings of the 4th International Workshop on Consuming Linked Data* (2013)
- Ruben Verborgh, Sam Coppens, Thomas Steiner, Joaquim Gabarró Vallés, Davy Van Deursen, and Rik Van de Walle. „Functional descriptions as the bridge between hypermedia apis and the semantic web". In: *Proceedings of the Third International Workshop on RESTful Design*. ACM. 2012, pp. 33–40
- Ruben Verborgh, Thomas Steiner, Davy Van Deursen, Sam Coppens, Erik Mannens, Rik Van de Walle, and Joaquim Gabarró Vallés. „RESTdescA FunctionalityCentered Approach to Semantic Service Description and Composition". In: *Proceedings of the Ninth Extended Semantic Web Conference* (2012)

- Ruben Verborgh, Thomas Steiner, Davy Van Deursen, Sam Coppens, Erik Mannens, Rik Van de Walle, and Joaquim Gabarró Vallés. „Integrating Data and Services through Functional Semantic Service Descriptions". In: *Proceedings of the W3C Workshop on Data and Services Integration* (2011)

Chapter 7

Remote Reasoning over SPARQL - From Query Federation to Distributed, Remote Reasoning

Until now, the SPARQL query language was restricted to simple entailment. Now SPARQL is being extended with more expressive entailment regimes. This allows to query over inferred, implicit knowledge. However, in this case the SPARQL endpoint provider decides which inference rules are used for its entailment regimes. In this chapter, we propose an extension to the SPARQL query language to support remote reasoning, in which the data consumer can define the inference rules. It will supplement the supported entailment regimes of the SPARQL endpoint provider with an additional reasoning step using the inference rules defined by the data consumer. At the same time, this solution offers possibilities to solve interoperability issues when querying remote SPARQL endpoints, which can support federated querying frameworks. These frameworks can then be extended to provide distributed, remote reasoning . This Chapter is thus an extension on top of Chapter 6

7.1 Introduction

Reasoning is one of main strengths of the Semantic Web. It infers logical consequences from graph structured RDF data based on inference rules. These inference rules are included in ontology languages (e.g., OWL2 [23]) or rule languages (e.g., N3 rules [16]). In order to produce

inferred triples, reasoners require, of course, access over the data. In a distributed environment as the Linked Open Data [14] cloud, this raises several concerns: First, all the data needs to be collected from different data sources and stored locally, before one can reason over the data. Second, when dealing with large distributed datasets, as is common in modern web applications, this centralisation of triples becomes problematic, since processor and memory consumption increase with the amount of triples. Third, in this Chapter, we argue that reasoning over large distributed datasets can be done more effectively if performed in a distributed, atomic way.

On the Semantic Web, RDF data can be accessed through a SPARQL endpoint [117]. Typically, triples can be retrieved by executing a SPARQL query. This process shows some strong similarities with reasoning, which are the following:

- both require the presence of the whole set of triples.
- both benefit from setting conditions and performing filters, to make the operation as precise as possible.
- both return a set of triples. In case of querying, this is the selected subset. In case of reasoning, these are the inferred triples.

We exploit the fact that the SPARQL infrastructure is already well deployed by extending the endpoint's functionality to support reasoning operations. With a single query, inference rules are sent to the endpoint and the inferred triples are returned.

In this Chapter, we introduce a novel approach to deal with the issues of reasoning over centralised triples, by allowing custom inference rules being executed over data stored on remote systems. We build on existing Semantic Web technology, by adding an extra layer to the SPARQL query language and reusing its endpoint infrastructure. We do not force SPARQL endpoints to support heavy reasoning tasks, but we leave open the possibility. They can decide to not support any reasoning. In this case, the reasoning could be performed by the SPARQL client. But by integrating this reasoning mechanism in the SPARQL protocol and syntax, SPARQL clients can be developed supporting the reasoning, e.g., a SPARQL client that relies on query rewriting for the inference rules supported by the OWL QL query language, and relying on client-side, OWL reasoning for the other inference rules. This kind of SPARQL client would not even need any adaptation to the existing SPARQL endpoints. Another example are SPARQL clients with a MapReduce-infrastructure to support the reasoning. In this Chapter, we only focus

on the extension for the SPARQL syntax and protocol. The SPARQL endpoints are free to decide what reasoning support they give, and SPARQL clients are free to decide how to schedule the reasoning (client vs. server) .

This Chapter is structured as follows. In Section 7.2, we start by discussing some related work.. Next, in Section 7.3, we motivate the benefits of remote reasoning and in Section 7.4 we discuss how we reused SPARQL to support it. Then, we describe two future use cases as a proof of concept in Section 7.6. Finally, in Section 7.7, we conclude with some future work and a final conclusion in Section 7.8.

7.2 Related work

Currently, the W3C SPARQL working group has proposed a recommendation for supporting entailment regimes [118] in SPARQL. The SPARQL 1.1 Query specification defines the evaluation of basic graph patterns by means of subgraph matching. This form of basic graph pattern matching is also called simple entailment. The proposed support of the entailment regimes will allow for retrieving solutions that implicitly follow from the queries graph. The proposed recommendation regimes are: RDF entailment, RDFS entailment, D-entailment, OWL2 RDF-based semantics entailment, OWL2 direct semantics entailment, and OWL QL core entailment. This way, a data provider can adapt its basic graph pattern matching algorithm to support one of the entailment regimes to support querying over entailed triples. This recommendation will allow data providers to publish SPARQL endpoints which support one of the proposed entailment regimes. In this solution, the data providers offer the ontology or rule-set used for reasoning and the reasoning actually happens on the level of basic graph pattern matching. In our solution, the end-user defines the ontology or rule-set used to reason over the queries graph. The reasoning in our solution happens after the basic graph pattern matching. Thus, we have a post-reasoning step, which supplements the pre-reasoning step offered by the entailment regime.

Related to our solution is *stream reasoning*. Event processing is concerned in timely detecting compound events in streams. Event processing is already capable of doing run-time analysis of the event streams. Stream reasoning on the other hand will allow event processors to combine background knowledge to entail extra knowledge. Event Processing SPARQL (EPSPARQL, [119]) is a new language for complex event and stream reasoning. Next to the new language for querying

streams of events, they also provide an execution model which derives information from streamed RDF events in real-time. The framework is extended based on event-driven backward chaining rules. These are logic rules that can thus be mixed with other background knowledge. Further investigation is needed here to support more expressive formalisms for stream reasoning, such as OWL [23] and its different profiles.

Distributed reasoning is also closely related to our framework. In such frameworks, reasoning is happening with multiple ontologies interrelated with semantic mappings on distributed data. LarKC (Large Knowledge Collider, [120]) and DRAGO (Distributed Reasoning Architecture for a Galaxy of Ontology, [121]) are such distributed reasoning frameworks. LarKC performs massive, distributed, and necessarily incomplete reasoning over web-scale knowledge sources. Massive inference is achieved by distributing problems across heterogeneous computing resources. LarKC is based on a pluggable architecture in which it is possible to exploit techniques and heuristics from diverse areas such as databases, machine learning, cognitive science, Semantic Web, and others. In DRAGO, reasoning is the result of a combination of semantic mappings of local reasoning chunks performed in a single OWL ontology. It provides reasoning services for multiple OWL ontologies, interconnected via C-OWL mappings. Future work for DRAGO involves extending the DRAGO framework to cope with distributed T-boxes and to support more expressive ontology mappings.

7.3 Remote reasoning

Remote Reasoning resides in the classic client-server architecture. It is a service allowing users to reason over its accessible data with a supplied rule set or OWL ontology. The reasoning can happen on three places: at the server, at the client or third party. This is demonstrated in Figure 7.1.

The benefits are three-fold. First, users can benefit from delegating some of the effort of reasoning to the server. Those reasoning tasks performed within the OWL QL space, can easily be performed by the server by query rewriting. This does not only result in faster reasoning times, but is also cost-efficient. Actually, in the case of query rewriting, the reasoning is done by query rewriting, typically at the client, followed by basic graph pattern matching at the server. Second, in many cases, a client only requires the reasoning results, i.e., the in-

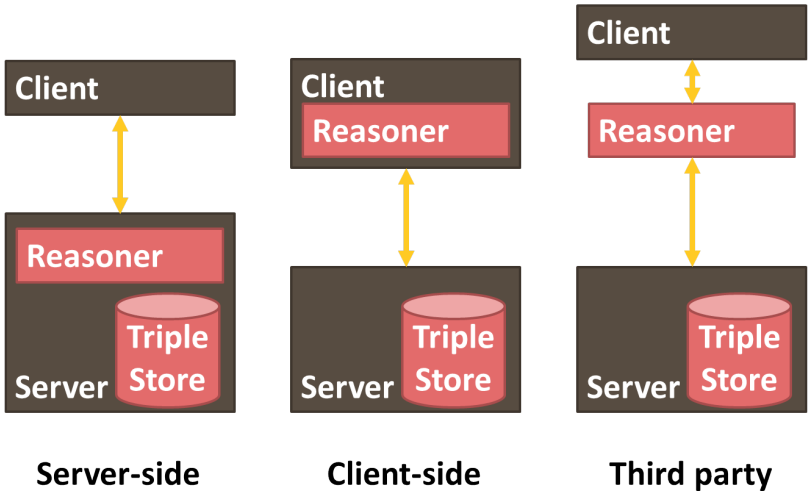

Figure 7.1: Client-Server Architectures for Remote Reasoning

ferred triples. Therefore, eliminating the harvesting and storing of data locally, drastically reduces overhead in terms of time and storage consumption. Also, data is not duplicated and synchronizing versions is therefore avoided. Third, the used inference rules are flexible, dynamic and defined by the client, not the server. At the moment SPARQL endpoints can already support some entailment regimes, but here the SPARQL endpoint providers decides which inference rules are used for reasoning. In many cases, the data consumer itself wants to define the inference rules to support its application.

In order to integrate remote reasoning on the Web, there are a few requirements. First, reasoning happens within the *constraints* of the server. Second, a *syntax* and *communication protocol* need to be defined to assure interoperability. In this Chapter we focus on this last requirement, which is discussed in the next Sections.

7.4 Extending SPARQL

In general, a reasoning cycle can be divided into five steps, as shown by LarKC:

1. Identification
2. Selection
3. Transformation
4. Reasoning
5. Decision

RDF Fragment 7.1: A REASON Query for executing N3 Rules declared in an external File over all triples

```
REASON <http://test.com/rules.n3>
OVER {
  ?s ?p ?o
} WHERE {
  ?s ?p ?o
}
```

SPARQL already covers the first three steps (CONSTRUCT queries can be considered as transformations). By extending the SPARQL query language, the fourth step can also be covered by SPARQL, allowing for remote reasoning.

7.4.1 The REASON query

Reasoning can be performed by executing a SPARQL query. Therefore, we introduce a new query form REASON equivalent to SELECT, CONSTRUCT, ASK, and DESCRIBE. The syntax consists of three main parts (as illustrated in RDF Fragment 7.1):

- the REASON keyword
- a *rule set* in an ontology or declarative language. This can either be a URL to a rule file, supplied between <>, or inline N3 rules between *{}*.
- the OVER and WHERE keyword combination which defines the triples to reason over. It is equivalent to the CONSTRUCT keyword in combination with its WHERE clause. Its content is placed between *{}*.

A REASON query returns a valid RDF graph, equivalent to the CONSTRUCT query. This graph includes only the inferred triples. We chose for the new SPARQL query form because:

- The structure of the REASON query form supports a selection step, as discussed in the context of LARKC [120] and Section 7.4. This way, we can do remote reasoning on a snapshot of the data. By doing this, the server keeps control over the memory consumption a certain reasoning task might require. The SPARQL endpoint provider can, e.g., require the selection step to trim down the graph to reason over 10.000 triples or to stop reasoning the moment it has inferred 10.000 triples, although the results can be incomplete.

- SPARQL allows for subqueries. Subqueries are a way to embed SPARQL queries within other queries, normally to achieve results which cannot otherwise be achieved, such as limiting the number of results from some sub-expression within the query. Thus, implementing the remote reasoning over SPARQL via a new query form allows to query the returned entailed triples, as will be shown in the next Section A.
- The combination of the REASON query form with a SERVICE extension of SPARQL provides a way to balance workload with federated querying, as explained in Section B.

A. Precise reasoning with nested queries

The SPARQL 1.1 specification allows subqueries and nesting of queries. This can be fully exploited to allow queries on the entailed triples. This enables fine-grained reasoning, which enables optimization of results and execution time. Examples of nested queries are given in RDF Fragment 7.2 and RDF Fragment 7.3.

RDF Fragment 7.2: Only the inferred Child Relationship of Jenna is selected.

```
SELECT ?child
WHERE {
  :Jenna :child ?child .
  {
    REASON {
      { ?x :parent ?y } => { ?y :child ?x } .
    }
    OVER {
       ?s :parent ?o .
    }
    WHERE {
    ?s a :Person; :parent ?o .
    }
  }
}
```

B. Balancing workload with Federated Querying

Federated querying is another possibility in SPARQL. It allows triples to be fetched from another endpoint, and include them in your query using the SERVICE element. When we include this into a REASON query, we can reason with one endpoint, over data from another. This is very

RDF Fragment 7.3: The worksWith Relationship is inferred only for Persons working at the University of Ghent

```
REASON {
  { ?x :knows ?y } => { ?y :knows ?x } .
}
OVER {?s :knows ?o}
WHERE
{
  CONSTRUCT {?s :workedWith ?o}
  WHERE {
    ?s :worksAt :UnivGhent.
    ?s :participatedInProject ?project.
    ?o :worksAt :UnivGhent.
    ?o :participatedInProject ?project.
  }
}
```

powerful, because it allows a high performance server to reason over remote data with a single query. Therefore, the biggest workload can be performed on the machine best suited for it. An example query is shown in RDF Fragment 7.4.

7.4.2 Ontology classification

The inference rules used for reasoning can come from OWL ontologies or declarative rules. Classification of the used inference rules will allow to select the appropriate reasoner for doing the inferencing. Today, highly optimised reasoners such as EYE[122], Pellet [123], FaCT++ [124], RacerPro [125] and HermiT [126] are able to classify many ontologies used in applications. The optimisations employed by these reasoners aim not only to improve performance on individual subsumption tests, but also to reduce the number of tests performed when classifying a given ontology. Such an ontology classification can also be a means to enforce a constraint of the server. The server can decide to only support OWL QL inference rules, because in this case the query can be rewritten and SPARQL can be exploited to the fullest, without actual generating inferred triples.

RDF Fragment 7.4: Example query showing how the data is first fetched from two endpoints and then used for reasoning

```
REASON {
  { ?x foaf:knows ?y } => { ?y foaf:knows ?x } .
}
OVER {
  :Jenna foaf:knows ?person .
        }
WHERE{
  {
  SERVICE <http://example.org/sparql> {
    :Jenna foaf:knows ?person .
  } } UNION {
  SERVICE <http://example2.org/sparql> {
    :Jenna foaf:knows ?person .
  } }
}
```

7.4.3 Handling triple selection validation for reasoning

When executing a REASON query, the triples to reason over are selected in the OVER clause. This principle holds a potential pitfall. When the selection lacks the necessary triples for the given inference rule set, the result will be empty. An example is given in RDF Fragment 7.5. Although this can not be considered an error, it needs to be discussed in the query specification. In the end, the user is made responsible for the triple selection in the OVER clause. When a result set turns up empty, it can have two reasons: (i) the inference rules lead to no inferred triples, or (ii) the selection of the triples to reason over was incomplete. Automatic validation is also possible using the aforementioned reasoners, but in combination with the entailment regimes, this becomes hard. A query can, at first sight, lead to no results and thus invalidation, but can also rely on the supported entailment regime of the SPARQL endpoint provider to provide the results.

RDF Fragment 7.5: Example Triple Selection in the OVER clause.

```
REASON {
  { ?x foaf:knows ?y }
    => { ?y foaf:knows ?x } .
}
OVER {
  :Jenna a foaf:Person .
}
WHERE {
  :Jenna ?p ?o .
}
```

7.5 A SPARQL reasoning endpoint

For the proof-of-concept, we have extended the Apache Jena ARQ[1] query engine to support remote reasoning. This engine is very popular and is often used for implementing SPARQL endpoints. By extending the ARQ library to support remote reasoning, existing SPARQL endpoints can easily be extended to support this remote reasoning. As explained in Section 7.4, we introduced a new SPARQL query form: the REASON query form. In our first prototype, this new SPARQL query form supports N3 rules. This way, we are already able to offer reasoning on top of SPARQL. For the future, special attention will go to OWL QL inference rules, because these can be rewritten. This way, the server impact can remain minimal. OWL QL seems to be the right fit for empowering SPARQL with reasoning capabilities.

7.6 Use cases

In this section, we discuss two possible use cases for remote reasoning: ontology interoperability and distributed reasoning. In both cases, the remote reasoning on top of SPARQL offers some possibilities, discussed in the next sections.

7.6.1 Ontology interoperability

Interoperability between different ontologies can be simplified by supplying mapping rules at query time similar to the approach followed by [127]. This allows a user to use a custom vocabulary in a SPARQL

[1] http://jena.apache.org/documentation/query/index.html

RDF Fragment 7.6: A query overcoming interoperability issues

```
SELECT ?name
WHERE {
  ?s a :Artist .
  {
    REASON {
      { ?x :role 'artist' } => { ?x a :Artist} .
    }
    OVER {
      ?s :role ?o
    }
    WHERE {
      ?s a foaf:Person; :role ?o
  }
}
```

query that differs from the one used in the endpoint. This is typically a problem for query federation frameworks, where different parts of a query are evaluated by different SPARQL endpoints. These SPARQL endpoints often use different vocabularies to describe their data. By including mapping rules, interoperability issues are avoided when querying. An example of a federated query, supporting ontology interoperability is shown in RDF Fragment 7.6.

7.6.2 Distributed reasoning

Many federated query frameworks rely on SPARQL to federate subqueries (parts of the incoming query) to the appropriate SPARQL endpoints and to aggregate all the results. These frameworks can already benefit from remote reasoning to solve interoperability issues between the contacted SPARQL endpoints. These federated querying frameworks could be extended with this SPARQL extension to become federated reasoning frameworks. The example shown in RDF Fragment 7.7 shows how this could work.

RDF Fragment 7.7: A subset from the reason results can be extracted using SPARQL features

```
SELECT ?artist
WHERE {
  ?artist a :Artist.
  { REASON {
      {?y dbpprop:artist ?x.}
      =>
      {?x a :Artist} .
    }
  OVER {
      ?artwork dbpprop:artist ?person .
    }
  WHERE {
      SERVICE <http://example.org/sparql> {
      ?person a foaf:Person.
      ?artwork dbpprop:artist ?person .
    }
  }
}
```

7.7 Future Work

Future work will first focus on the following things:

- We want to add support for other rule languages besides N3. Special focus will go to inference rules belonging to the OWL QL profile.
- For the OWL QL inference rules, reasoning can be replaced by query rewriting. These conversions will be investigated for integration into our solution.
- For other inference rules, dedicated reasoners can be selected. Here, the classification comes into play. Dedicated reasoners perform much faster then general purpose reasoners. Thus a good classification can increase performance by selecting dedicated reasoners or a combination of dedicated reasoners. In order to exploit this feature, we will need to be able to split up an ontology into different parts, each part being able of taking care of a specific dedicated reasoner. After that, we need to plan how these dedicated reasoners will exchange their results, to form the overall result of the ontology. Thus research here will focus

on three areas: classification of the ontology's inference rules, split up of the ontology, according to the classification of the inference rules, and finally planning of the reasoning cycle over the dedicated reasoners to infer the overall result of reasoning with data over the whole ontology.

A second objective for the future will be the integration of remote reasoning via SPARQL into federated querying frameworks to build a distributed reasoner, as shown in Section 7.6. If we can distribute incoming queries over disparate SPARQL endpoints and we can distribute the reasoning at the same time to the external endpoints, we achieve the goal of remote reasoning over distributed data.

7.8 Conclusion

Distributed reasoning is becoming more important nowadays, because of the distributed characteristic of Linked Open Data. In this chapter, we have proposed an extension to the SPARQL query language to support remote reasoning. This extension will support two use cases: ontology interoperability, e.g., when federating queries, and distributed reasoning. In the future, reasoning on distributed data sources will become more important and this extension to SPARQL can be a building block to facilitate distributed reasoning. By extending SPARQL, the selection of data to reason over is incorporated into the remote reasoning framework we propose. This is a very important feature to restrict the amount of data to reason over, because it allows to reason on certain snapshots of distributed data, e.g., versions.

The author's work on federated and distributed reasoning, and in general federated querying led to the following publications.

- Sam Coppens, Miel Vander Sande, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. „Remote Reasoning over SPARQL". in: *Proceedings of the 6th Workshop on Linked Data on the Web* (2013)
- Sam Coppens, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. „Querying Distributed Linked Data - Query Federation using the Provenance of owl:sameAs Links". In: *Journal of Data and Information Quality* (2014)
- Sam Coppens, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. „Querying the Linked Data Graph using owl: sameAs Provenance". In: *Proceedings of the 16th International Conference on Model Driven Engineering Languages and Systems* (2013)

Chapter 8

Conclusions

The topic of the dissertation is provenance information for the Semantic Web. Provenance has been researched from its different dimensions, i.e., representation, generation, dissemination and consumption. Each chapter in this dissertation tackled one of these areas of research for provenance information. In Chapter 1, we introduced provenance information, and its different application in different domains, and a core data model for representing provenance information. Next, we also gave an overview of the foundations of the Semantic Web. The main research topics of provenance information can be divided into four categories: provenance modeling, provenance generation, provenance dissemination and provenance consumption. The remainder of the dissertation touches every domain of the provenance research. In Chapter 2, we focus on the modeling and representation of provenance information. In Chapter 3, we propose a framework for the generation of provenance information. Chapter 4 proposes a way of disseminating provenance information on the Web, and in Chapter 5, we show how provenance information can be used. In the last two chapters, Chapter 6 and Chapter 7, we introduce some extensions on the previous chapters. Chapter 6 proposes a semantic workflow engine that could generate provenance information and that could serve as back-end for the long-term preservation infrastructure, described in Chapter 3. The last chapter, Chapter 7, describes an extension to the SPARQL protocol that could extend the distributed querying framework, detailed in Chapter 5, to become a distributed, remote reasoning framework. In what follows, we show for each of our handled topics what the novel contributions are that were introduced.

8.1 Provenance Representation: PREMIS OWL

In the domain of provenance representation, we described PREMIS OWL in Chapter 2. PREMIS OWL is a semantic data model for preservation metadata. Preservation metadata is a domain-specific provenance model that extends provenance information with structural metadata, technical metadata and rights metadata to support the long-term preservation of resources. The main contribution of PREMIS OWL is that is not only describes the structure of the preservation metadata, but also its semantics. At the same time, this model can be used to disseminate the preservation metadata as Linked Open Data on the Web. PREMIS OWL also introduces 24 preservation vocabularies that were developed by the Library of Congress. This feature makes the preservation metadata interoperable amongst the archives. Each archive has its specific preservation processes in place to fulfill its tasks. These specific processes need to be described in the preservation metadata. Each archive can describe its preservation processes using its own SKOS vocabularies. To be conform to PREMIS OWL, these SKOS vocabularies need to be linked to the 24 preservation vocabularies introduced in PREMIS OWL. This will make the preservation metadata interoperable. This feature is needed as archived objects are more and more being exchanged among archives across the world.

8.2 Provenance Generation: Archipel

Considering the provenance generation, we introduced Archipel, described in Chapter 3. Archipel is a distributed, long-term preservation infrastructure. This is an infrastructure that will keep all the preserved data intact and interpretable over time. Our platform has all the necessary processes to guarantee the long-term preservation of the incoming data. For the platform, we used a service-oriented architecture, where all the preservation services are implemented as services on an enterprise service bus.
All the stored data are stored inside packages, to obey the OAIS guidelines. These packages contain descriptive metadata, referenced multimedia files and preservation metadata. This way, a package contains all the necessary information for its preservation purposes. These packages (BagIt packages) are stored in the cloud. This means they are replicated among different storage devices in a cloud environment. The metadata of the packages are also stored in a triple store for dissemination as

Linked Open Data. The infrastructure uses PREMIS OWL to describe its preservation metadata and disseminates all its resources as Linked Open Data, not only the descriptive metadata, but also the preservation metadata. The preservation metadata allows an end-user to browse to previous versions of the data. The infrastructure provides all the preservation metadata autonomously, including the technical metadata. For this technical metadata, the system relies on technical registries, published as Linked Open Data. Essential technical information, such as file format information, is also stored locally, such that the long-term preservation infrastructure can stand on its own at all times.

8.3 Provenance Dissemination: Memento and Provenance

When preserving information for the long-term and publishing it as Linked Open Data at the same time using persistent identifiers some problems arise, which are tackled in Chapter 4. First of all, using the persistent identifiers, we want to provide also all the different versions over time of a certain resource. A system as Memento, using datetime content negotiation, offers a solution for this. At the same time, it solves also the issue of the temporarlity of the enrichments when publishing Linked Open Data and using persistent identifiers. For instance, if a record of the Gulf War refers to a record of the president of the US, it must refer to the record of George W. Bush, not the current president. Datetime content negotiation solves this issue. Next we extended the Memmento datetime content negotiation to include also HTTP link headers for the provenance information of the different versions of the published data. The provenance information forms the glue between the different versions of the data and this offers another mechanism to browse the different versions of the data. Thus, not only via the datetime content negotiation, but also via the provenance information. In this framework, provenance information is also considered versionable or datetime content negotiable. Thus, this allows even to publish the provenance information using a persistent URI.

8.4 Provenance Consumption: Query Federation

In Chapter 5, we show how provenance information can even help in query distribution. The provenance of the owl:sameAs links provide all the information for query distribution. The provenance information of an owl:sameAs link can be represented as an interlinking rule. From this rule, we can extract the used SPARQL endpoint and even vocabulary mappings used for querying the remote dataset. This information is being reused for distributing the incoming queries. For this, we developed an extension to ARQ. ARQ is a popular SPARQL query processing library. Our extension can implicitly extract all the needed information to support its query federation. The ARQ extension can be fed with the interlinking rules you used as a dataset provider for publishing your dataset as Linked Open Data. The extension can also be fed with SILK configuration files from which the needed information can be extracted. SILK is a popular interlinking framework. With our extended ARQ library, every SPARQL endpoint can be turned into a query federation endpoint with minimal effort.

8.5 Self-Sustaining Platforms: Semantic Workflow Engine

Chapter 6 describes a semantic workflow engine. In fact, it is a distributed reasoning framework on top of which a semantic workflow engine is implemented. Nowadays, the intelligence of a systems is often scattered among the software, the services it uses, and the workflows in which the services are used. With this system, we want to provide a system that aggregates as much of intelligence as possible. For this our system relies on a three-step rule-reasoning cycle. During each step, a reasoning task is performed, described in a rule file. By doing this, we aggregate as much as possible the intelligence in the three rule files. At the same time, these rule files can be changed at run-time with a direct effect. This means we can easily extend the workflow engine with new services, or even new workflows at run-time, e.g., the inclusion of a new sensor that steers the workflow in process control systems. A third benefit of the workflow engine is that workflows are first composed in terms of functionalities. Later on these functionalities are replaced by services providing these functionalities. This benefits the management of the services and the workflows that used these services.Services can

be updated with new functionalities or new services can be developed providing the same functionality of an existing service. Our workflow engine can handle all these changes, because it first composes a workflow in terms of functionalities. A last benefit of this workflow engine is that the three steps of the reasoning cycle can be scheduled on a grid, making the workflow engine very scalable and suitable for smart city solutions.

8.6 Remote Reasoning over SPARQL

In the last chapter, Chapter 7, we proposed an extension to the SPARQL query language and protocol to support remote reasoning. Before the SPARQL 1.1 specification, the SPARQL query language was restricted to simple entailment. In the new SPARQL 1.1 specification, SPARQL is being extended with more expressive entailment regimes. This allows to query over inferred, implicit knowledge. However, in this case the SPARQL endpoint provider decides which inference rules are used for its entailment regimes. In the solution provided in Chapter 7 the client of the SPARQL endpoint can decide which inference rules it want to use for its entailment regimes. Before this solution, the client had to fetch the remote knowledge base first locally in order to reason over that knowledge base. Now, it can reason over that knowledge base remotely at query execution time. At the same time, this solution offers possibilities to solve interoperability issues when querying remote SPARQL endpoints, which can support federated querying frameworks, as the one described in Chapter 6. These frameworks can then be extended to provide distributed, remote reasoning capabilities.

Appendix A

Example RDFS Model

RDF Fragment A.1: Example RDFS model

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix voc: <http://example.org/voc/vocabulary#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

voc:Artist                    rdf:type        rdfs:Class;
                              rdfs:label      "artist";
                              rdfs:comment    ""’"’This class
                                  is used for declaring
                                              artists in your
                                                  RDF model.""’
                                                  "’.

voc:Artwork                   rdf:type        rdfs:Class;
                              rdfs:label      "artwork";
                              rdfs:comment    ""’"’This class
                                  is used for declaring
                                              artworks in your
                                                  RDF model.""’
                                                  "’.

voc:ContemporaryArtwork       rdf:type        rdf:Class;
                              rdfs:label      ""’"’contemporary
                                   artwork"’"’";
                              rdfs:comment    ""’"’This class
                                  describes a
                                              contemporary
                                                  artwork""’"’;
                              rdfs:subClassOf voc:Artwork.

voc:ContemporaryArtist        rdf:type        rdf:Class;
```

```
                        rdfs:label      ""'"'contemporary
                            artist""'"'";
                        rdfs:comment    ""'"'This class
                            describes a
                                        contemporary
                                            artist"'"'".
                        rdfs:subClassOf voc:Artist.

voc:title               rdf:type        rdf:Property;
                        rdfs:label      "title";
                        rdfs:comment    ""'"'This
                            property gives the title of
                                        an artwork""'"';
                        rdfs:domain     voc:Artwork;
                        rdfs:range      rdfs:Literal.

voc:description         rdf:type        rdf:Property;
                        rdfs:label      "description";
                        rdfs:comment    ""'"'This
                            property gives the
                            description
                                        of an artwork"'"'
                                            ";
                        rdfs:domain     voc:Artwork;
                        rdfs:range      rdfs:Literal.

voc:date                rdf:type        rdf:Property;
                        rdfs:label      "date";
                        rdfs:comment    ""'"'This
                            property gives the date
                                        the artwork was
                                            created""'"';
                        rdfs:domain     voc:Artwork;
                        rdfs:range      xsd:dateTime.

voc:creator             rdf:type        rdf:Property;
                        rdfs:label      "creator";
                        rdfs:comment    "`"'"This
                            property links an artist
                                        to the artwork as
                                             a creator"'"
                                            '";
                        rdfs:domain     voc:Artwork;
                        rdfs:range      voc:Artist.
```

Appendix B

OWL Model

Equality:

- owl:equivalentClass: to denote that a certain class is the same as another class
- owl:equivalentProperty: to denote a certain property means the same as another property
- owl:sameAs: to denote certain individuals are actually the same
- owl:differentFrom: to express two individuals are different
- owl:AllDifferent: which denotes that a number of individuals are mutually distinct.
- owl:disjointWith: Classes may be stated to be disjoint from each other.

Property Characteristics:

- owl:inverseOf: expresses two properties are the inverse of each other
- owl:TransivityProperty: a class for transitive properties), owl:SymmetricProperty (a class for symmetrical properties
- owl:FunctionalProperty: if a property is a owl:FunctionalProperty, then it has no more than one value for each individual
- owl:InverseFunctionalProperty: If a property is inverse functional, then the inverse of the property is functional.

Property Restrictions:

- owl:allValuesFrom: means that this property on this particular class has a local range restriction associated with it.
- owl:someValuesFrom: a restriction on a property that at least one of its values is of a certain type.
- owl:hasValue: a property can be required to have a certain individual as a value.

Restricted Cardinality:

- owl:minCardinality: denotes the number of different values a certain property may have with respect to a certain class, e.g., an artwork has at least one creator
- owl:maxCardinality: expresses the maximum number of different values a certain property may have with respect to a certain class.
- owl:cardinality: combines the two previous cardinality restrictions.

Class Intersection:

- owl:intersectionOf: defines a class as the intersection of two classes, e.g., one can define the class of employed persons as the intersection of a person class and a employed things class.

- owl:unionOf: defines a class as the union of two classes.
- owl:complementOf: define a class as the complement of another class. e.g., females and males.

Class Enumeration:

- owl:oneOf: Classes can be described by enumeration of the individuals that make up the class. The members of the class are exactly the set of enumerated individuals; no more, no less.

Appendix C

OWL Example Ontology

RDF Fragment C.1: Example OWL model

```
@prefix : <http://example.org/voc/vocabulary#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix voc: <http://example.org/voc/vocabulary#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@base <http://example.org/voc/vocabulary#> .

<http://example.org/voc/vocabulary#> rdf:type owl:Ontology .

:Artwork rdf:type owl:Class ;
     rdfs:subClassOf [ rdf:type owl:Restriction ;
                       owl:onProperty :date ;
                       owl:qualifiedCardinality "1"^^xsd:
                           nonNegativeInteger ;
                       owl:onDataRange xsd:string
                     ] ,
                     [ rdf:type owl:Restriction ;
                       owl:onProperty :creator ;
                       owl:onClass :Artist ;
                       owl:minQualifiedCardinality "1"^^xsd:
                           nonNegativeInteger
                     ] ,
                     [ rdf:type owl:Restriction ;
                       owl:onProperty :title ;
                       owl:minQualifiedCardinality "1"^^xsd:
                           nonNegativeInteger ;
                       owl:onDataRange xsd:string
```

```
                    ] .

:Artist rdf:type owl:Class ;
     rdfs:subClassOf foaf:Person ,
                    [ rdf:type owl:Restriction ;
                      owl:onProperty :created ;
                      owl:onClass :Artwork ;
                      owl:minQualifiedCardinality "1"^^xsd:
                          nonNegativeInteger
                    ] .

:ContemporaryArtwork rdf:type owl:Class ;
     rdfs:subClassOf :Artwork ,
                    [ rdf:type owl:Restriction ;
                      owl:onProperty :date ;
                      owl:someValuesFrom
                               [ rdf:type rdfs:Datatype ;
                                 owl:onDatatype xsd:dateTime ;
                                 owl:withRestrictions ( [ xsd:
                                     minExclusive "1945-01-01
                                     T00:00:00"])
                                ]
                    ] .

:ContemporaryArtist rdf:type owl:Class ;
     rdfs:subClassOf :Artist ,
                    [ rdf:type owl:Restriction ;
                      owl:onProperty :created ;
                      owl:onClass :ContemporaryArtwork ;
                      owl:minQualifiedCardinality "1"^^xsd:
                          nonNegativeInteger
                    ] .

:created rdf:type owl:ObjectProperty ;
         rdfs:domain :Artist ;
         rdfs:range :Artwork ;
         owl:inverseOf :creator .

:creator rdf:type owl:ObjectProperty ;
          rdfs:subPropertyOf dc:creator;
         rdfs:range :Artist ;
         rdfs:domain :Artwork .

:date rdf:type owl:DatatypeProperty ;
      rdfs:subPropertyOf dc:date;
      rdfs:domain :Artwork ;
      rdfs:range xsd:dateTime .

:description rdf:type owl:DatatypeProperty ;
```

```
            rdfs:subPropertyOf dc:description;
            rdfs:domain :Artwork ;
            rdfs:range xsd:string .

:title rdf:type owl:DatatypeProperty ;
       rdfs:subPropertyOf dc:title;
       rdfs:domain :Artwork ;
       rdfs:range xsd:string .
```

Publications

Journal papers

- Sam Coppens, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. „PREMIS OWL - A Semantic Long-Term Preservation Model". In: *International Journal on Digital Libraries* (2014)
- Sam Coppens, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. „Self-Sustaining Platforms - Grid Reasoning". In: *Transactions on Autonomous and Adaptive Systems* (2014)
- Erik Mannens, Sam Coppens, Toon De Pessemier, Hendrik Dacquin, Davy Van Deursen, Robbie De Sutter, and Rik Van de Walle. „Automatic news recommendations via aggregated profiling". In: *Multimedia Tools and Applications* (2013), pp. 1–19
- Sam Coppens, Erik Mannens, Toon De Pessemier, Kristof Geebelen, Hendrik Dacquin, Davy Van Deursen, and Rik Van de Walle. „Unifying and targeting cultural activities via events modelling and profiling". In: *Multimedia Tools and Applications* 57.1 (2012), pp. 199–236
- Toon De Pessemier, Sam Coppens, Kristof Geebelen, Chris Vleugels, Stijn Bannier, Erik Mannens, Kris Vanhecke, and Luc Martens. „Collaborative recommendations with content-based filters for cultural activities via a scalable event distribution platform". In: *Multimedia Tools and Applications* 58.1 (2012), pp. 167–213
- Erik Mannens, Ruben Verborgh, Seth Van Hooland, Laurence Hauttekeete, Tom Evens, Sam Coppens, and Rik Van de Walle. „On the Origin of Metadata". In: *Information* 3.4 (2012), pp. 790–808
- Sam Coppens, Jan Haspeslagh, Patrick Hochstenbach, Erik Mannens, Rik Van de Walle, and Inge Van Nieuwerburgh. „Metadatastandaarden, Dublin Core en het gelaagd metadatamodel". eng. In:

ed. by Stoffel Debuysere, Dries Moreels, Rik Van de Walle, Inge Van Nieuwerburgh, and Jeroen Walterus. Bewaring en ontsluiting van multimediale data in Vlaanderen : perspectieven op audiovisueel erfgoed in het digitale tijdperk. Lannoo Campus, 2010, pp. 46–62. ISBN: 9789020989441

- Erik Mannens, Sam Coppens, and Rik Van de Walle. „Semantic BRICKS for performing arts archives and disseminiation". In: *IASA JOURNAL* 35 (2010), pp. 40–49
- Sam Coppens, Erik Mannens, and Rik Van de Walle. „Disseminating heritage records as linked open data". In: *International Journal of Virtual Reality* 8.3 (2009), pp. 39–44

Books

- Paul Bastijns, Sam Coppens, Siska Corneillie, Patrick Hochstenbach, Erik Mannens, and Liesbeth Van Melle. „(Meta)datastandaarden voor digitale archieven". dut. In: (2009). Ed. by Rik Van de Walle and Sylvia Van Peteghem, p. 199. URL: `http://www.archive.org/details/metadatastandaardenVoorDigitaleArchieven`

Book chapters

- Rik Van de Walle, Sam Coppens, and Erik Mannens. „Een gelaagd semantisch metadatamodel voor langetermijnarchivering". In: *BIBLIOTHEEK-EN ARCHIEFGIDS* 84.5 (2009), pp. 17–22
- Sam Coppens, Erik Mannens, and Rik Van de Walle. „Semantic BRICKS for performing arts archives and dissemination". eng. In: ed. by Tom Evens and Dries Moreels. Access to Archives of Performing Arts Multimedia. Vlaams Theater Instituut (VTI), 2009, pp. 121–141. ISBN: 9789074351386
- Stijn Notebaert, Jan De Cock, Sam Coppens, Erik Mannens, Rik Van de Walle, Marc Jacobs, Joeri Barbarien, and Peter Schelkens. „Digital recording of performing arts: formats and conversion". eng. In: ed. by Tom Evens and Dries Moreels. Access to Archives of Performing Arts Multimedia. Vlaams Theater Instituut (VTI), 2009, pp. 95–119. ISBN: 9789074351386

Standardisation activities

W3C

- Luc Moreau, Paolo Missier, Khalid Belhajjame, Reza B'Far, James Cheney, Sam Coppens, Stephen Cresswell, Yolanda Gil, Paul Groth, Graham Klyne, Timothy Lebo, Jim McCusker, Simon Miles, James Myers, Satya Sahoo, and Curt Tilmes. *PROV-DM: The PROV Data Model*. Tech. rep. URL: http://www.w3.org/TR/prov-dm/
- Sam Coppens and Tom De Nies. *PROV-Dictionary: Modeling Provenance for Dictionary Data Structures*. Tech. rep. 2013. URL: http://www.w3.org/TR/prov-dictionary/
- Yolanda Gil, James Cheney, Paul Groth, Olaf Hartig, Simon Miles, Luc Moreau, Paulo Pinheiro Da Silva, Sam Coppens, Daniel Garijo, JM Gomez, et al. „Provenance XG Final Report". In: (Dec. 2010). URL: http://www.w3.org/2005/Incubator/prov/XGR-prov/
- Satya Sahoo, Paul Groth, Olaf Hartig, Simon Miles, Sam Coppens, James Myers, Yolanda Gil, Luc Moreau, Jun Zhao, Michael Panzer, et al. *Provenance vocabulary mappings*. Tech. rep. 2010. URL: http://www.w3.org/2005/Incubator/prov/wiki/Provenance_Vocabulary_Mappings

Library of Congress

- Sam Coppens, Sebastien Peyrard, Rebecca Guenther, Keving Ford, and Tom Creighton. *PREMIS OWL ontology 2.2*. Tech. rep. 2013. URL: http://www.loc.gov/standards/premis/ontology-announcement.html
- Sam Coppens, Sebastien Peyrard, Rebecca Guenther, Keving Ford, and Tom Creighton. *PREMIS OWL ontology 2.1*. Tech. rep. 2011. URL: http://www.loc.gov/standards/premis/owlOntology-announcement.html

Papers in conference proceedings

2013

- Sam Coppens, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. „Querying the Linked Data Graph using owl: sameAs Provenance". In: *Proceedings of the 16th International Conference on Model Driven Engineering Languages and Systems* (2013)
- Sam Coppens, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. „Self-Sustaining Platforms: a semantic workflow engine". In: *Proceedings of the 4th International Workshop on Consuming Linked Data* (2013)
- Miel Vander Sande, Sam Coppens, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. „Adding Time to Linked Data: A Generic Memento proxy through PROV". in: *Poster and Demo Proceedings of the 12th International Semantic Web Conference* (2013)
- Tom De Nies, Sara Magliacane, Ruben Verborgh, Sam Coppens, Paul Groth, Erik Mannens, and Rik Van de Walle. „Git2PROV: Exposing Version Control System Content as W3C PROV". in: *Poster and Demo Proceedings of the 12th International Semantic Web Conference* (2013)
- Sam Coppens, Miel Vander Sande, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. „Remote Reasoning over SPARQL". in: *Proceedings of the 6th Workshop on Linked Data on the Web* (2013)
- Laurens De Vocht, Sam Coppens, Ruben Verborgh, Miel Vander Sande, Erik Mannens, and Rik Van de Walle. „Discovering Meaningful Connections between Resources in the Web of Data". In: *Proceedings of the 6th Workshop on Linked Data on the Web* (2013)
- Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Sam Coppens, Erik Mannens, and Rik Van de Walle. „R&Wbase: Git for triples". In: *Proceedings of the 6th Workshop on Linked Data on the Web* (2013)

2012

- Sam Coppens, Ruben Verborgh, Miel Vander Sande, Davy Van Deursen, Erik Mannens, and Rik Van de Walle. „A truly Read-Write Web for machines as the next-generation Web?" In: *Proceedings of the SW2012 workshop: What will the Semantic Web look like 10years from now* (2012)

- Miel Vander Sande, Sam Coppens, Davy Van Deursen, Erik Mannens, and Rik Van de Walle. „The Terminatorś origins or how the Semantic Web could endanger Humanity". In: ()
- Miel Vander Sande, Ruben Verborgh, Sam Coppens, Tom De Nies, Pedro Debevere, Laurens De Vocht, Pieterjan De Potter, Davy Van Deursen, Erik Mannens, and Rik Van de Walle. „Everything is Connected: Using Linked Data for Multimedia Narration of Connections between Concepts." In: *International Semantic Web Conference (Posters & Demos)*. 2012
- Tom De Nies, Evelien D'heer, Sam Coppens, Davy Van Deursen, Erik Mannens, and Rik Van de Walle. „Bringing Newsworthiness into the 21st Century". In: *11th International Semantic Web Conference (ISWC-2012)*. 2012, pp. 106–117
- Ruben Verborgh, Sam Coppens, Thomas Steiner, Joaquim Gabarró Vallés, Davy Van Deursen, and Rik Van de Walle. „Functional descriptions as the bridge between hypermedia apis and the semantic web". In: *Proceedings of the Third International Workshop on RESTful Design*. ACM. 2012, pp. 33–40
- Tom De Nies, Sam Coppens, Davy Van Deursen, Erik Mannens, and Rik Van de Walle. „Automatic discovery of high-level provenance using semantic similarity". In: *Provenance and Annotation of Data and Processes*. Springer Berlin Heidelberg, 2012, pp. 97–110
- Ruben Verborgh, Thomas Steiner, Davy Van Deursen, Sam Coppens, Erik Mannens, Rik Van de Walle, and Joaquim Gabarró Vallés. „RESTdescA FunctionalityCentered Approach to Semantic Service Description and Composition". In: *Proceedings of the Ninth Extended Semantic Web Conference* (2012)
- Erik Mannens, Sam Coppens, Ruben Verborgh, Laurence Hauttekeete, Davy Van Deursen, and Rik Van de Walle. „Automated Trust Estimation in Developing Open News Stories: Combining Memento & Provenance". In: *Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual*. IEEE. 2012, pp. 122–127

2011

- Sam Coppens, Erik Mannens, Davy Van Deursen, Patrick Hochstenbach, Bart Janssens, and Rik Van de Walle. „Publishing provenance information on the web using the Memento datetime content negotiation". In: *WWW2011 workshop on Linked Data on*

the Web (LDOW 2011). Vol. 813. 2011, pp. 6–15

- Sam Coppens, Erik Mannens, Raf Vandesande, and Rik Van de Walle. „Digital long-term preservation, provenance and linked open data". In: *2011 European Library Automation Group conference (ELAG 2011): It's the context, stupid!* 2011
- Sam Coppens, Erik Mannens, and Rik Van de Walle. „What about the provenance of media archives: a tale of time-contextualisation". In: (2011)
- Ruben Verborgh, Thomas Steiner, Davy Van Deursen, Sam Coppens, Erik Mannens, Rik Van de Walle, and Joaquim Gabarró Vallés. „Integrating Data and Services through Functional Semantic Service Descriptions". In: *Proceedings of the W3C Workshop on Data and Services Integration* (2011)
- Toon De Pessemier, Sam Coppens, Erik Mannens, Simon Dooms, Luc Martens, and Kristof Geebelen. „An event distribution platform for recommending cultural activities". In: *7th International Conference on Web Information Systems and Technologies (WEBIST-2011)*. Ghent University, Department of Information technology. 2011, pp. 231–236

2010

- Sam Coppens, Erik Mannens, and Rik Van de Walle. „PREMIS OWL binding to workflow engine for digital long-term preservation". In: (2010)
- Erik Mannens, Sam Coppens, Rik Van de Walle, Laurence Hauttekeete, and Robbie De Sutter. „Cloud-computing approach to sustainable media archives". In: (2010)
- Erik Mannens, Sam Coppens, and Rik Van de Walle. „Network-centric approach to sustainable digital archives". In: (2010)
- Erik Mannens, Sam Coppens, Toon De Pessemier, Hendrik Dacquin, Davy Van Deursen, and Rik Van de Walle. „Automatic news recommendations via profiling". In: *Proceedings of the 3rd international workshop on Automated information extraction in media production*. ACM. 2010, pp. 45–50

2009

- Sam Coppens, Erik Mannens, Tom Evens, Laurence Hauttekeete, and Rik Van de Walle. „Digital long-term preservation using a layered semantic metadata schema of PREMIS 2.0". In: *Cultural heritage on line, Florence, 15th-16th December* (2009)
- Sam Coppens, Erik Mannens, and Rik Van de Walle. „Semantic BRICKS for performing arts archives and dissemination". eng. In: ed. by Tom Evens and Dries Moreels. Access to Archives of Performing Arts Multimedia. Vlaams Theater Instituut (VTI), 2009, pp. 121–141. ISBN: 9789074351386
- Erik Mannens, Sam Coppens, Toon De Pessemier, Kristof Geebelen, Hendrik Dacquin, and Rik Van de Walle. „Unifying and targeting cultural activities via events modelling and profiling". In: *Proceedings of the 1st ACM international workshop on Events in multimedia*. ACM. 2009, pp. 33–40

References

References

[1] *The Semantic Web: Research and Applications - 8th Extended Semantic Web Conference, ESWC 2011, Heraklion, Crete, Greece, May 29-June 2, 2011, Proceedings, Part I*. Vol. 6643. Lecture Notes in Computer Science. Springer, 2011. ISBN: 978-3-642-21033-4.

[2] Berners-Lee, T. „Oh Yeah Button." In: *W3C Design Issues* (2006). URL: http://www.w3.org/DesignIssues/UI.html#ohyeah.

[3] Moreau, L.; Clifford, B.; Freire, J.; Futrelle, J.; Gil, Yo.; Groth, P.; Kwasnikowska, N.; Miles, S.; Missier, P.; Myers, J.; Plale, B.; Simmhan, Y.; Stephan, E. and Van den Bussche, J. „The Open Provenance Model core specification (v1.1)". In: *Future Gener. Comput. Syst.* 27.6 (June 2011), pp. 743–756. ISSN: 0167-739X. DOI: 10.1016/j.future.2010.07.005. URL: http://dx.doi.org/10.1016/j.future.2010.07.005.

[4] Sahoo, S. S.; Weatherly, D. B.; Mutharaju, R.; Anantharam, P.; Sheth, A. and Tarleton, R. L. „Ontology-Driven Provenance Management in eScience: An Application in Parasite Research". In: *Proceedings of the Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009 on On the Move to Meaningful Internet Systems: Part II*. OTM '09. Vilamoura, Portugal: Springer-Verlag, 2009, pp. 992–1009. ISBN: 978-3-642-05150-0. DOI: 10.1007/978-3-642-05151-7_18. URL: http://dx.doi.org/10.1007/978-3-642-05151-7_18.

[5] Ciccarese, P.; Wu, E.; Wong, G.; Ocana, M.; Kinoshita, J.; Ruttenberg, A. and Clark, T. „The SWAN biomedical discourse ontology". In: *J. of Biomedical Informatics* 41.5 (Oct. 2008), pp. 739–751.

ISSN: 1532-0464. DOI: 10.1016/j.jbi.2008.04.010. URL: http://dx.doi.org/10.1016/j.jbi.2008.04.010.

[6] Hartig, O. and Zhao, J. „Publishing and Consuming Provenance Metadata on the Web of Linked Data". In: *Provenance and Annotation of Data and Processes - Third International Provenance and Annotation Workshop, IPAW 2010, Troy, NY, USA, June 15-16, 2010. Revised Selected Papers*. Ed. by McGuinness, D. L.; Michaelis, J. and Moreau, L. Vol. 6378. Lecture Notes in Computer Science. Springer, 2010, pp. 78–90. ISBN: 978-3-642-17818-4. DOI: http://dx.doi.org/10.1007/978-3-642-17819-1_10.

[7] PREMIS working group. *PREMIS Data Dictionary for Preservation Metadata - version 2.0*. Available at http://www.loc.gov/standards/premis/v2/premis-2-0.pdf. Mar. 2008.

[8] Carroll, J. J.; Bizer, C.; Hayes, P. and Stickler, P. *Semantic Web Publishing Vocabulary*. http://www.w3.org/2004/03/trix/swp-2/. Nov. 4, 2004.

[9] Tunnicliffe, S. and Davis, I. *Changeset*. http://vocab.org/changeset/schema.html. May 18, 2009.

[10] Gil, Y.; Miles, S.; Belhajjame, K.; Deus, H.; Garijo, D.; Klyne, G.; Missier, P.; Soiland-Reyes, S. and Zednik, S. *PROV Model Primer*. Tech. rep. W3C, 2012. URL: http://www.w3.org/TR/prov-primer/.

[11] Internet Engineering Task Force. *RFC 3986: Uniform Resource Identifier (URI) – Generic Syntax*. Available at http://tools.ietf.org/html/rfc3986. 2005.

[12] Internet Engineering Task Force. *RFC 2616: HyperText Transfer Protocol – HTTP/1.1*. Available at http://www.ietf.org/rfc/rfc2616.txt. 1999.

[13] Pemberton, S., ed. *XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)*. W3C Recommendation. Available at http://www.w3.org/TR/xhtml1/. World Wide Web Consortium, Aug. 2002.

[14] Bizer, C.; Heath, T.; Idehen, K. and Berners-Lee, T. „Linked Data on the Web". In: *Proceedings of the 17th International World Wide Web Conference – LDOW Workshop*. Beijing, China, Apr. 2008, pp. 1265–1266.

[15] Beckett, D., ed. *RDF/XML Syntax Specification (Revised)*. W3C Recommendation. Available at `http://www.w3.org/TR/rdf-syntax-grammar/`. World Wide Web Consortium, Feb. 2004.

[16] Berners-Lee, T. and Connolly, D. *Notation 3*. Available at `http://www.w3.org/DesignIssues/Notation3`. 2006.

[17] Beckett, D. and Berners-Lee, T. *Turtle - Terse RDF Triple Language, W3C Team Submission*. See: http://www.w3.org/TeamSubmission/turtle/. 2008. URL: `http://www.w3.org/TeamSubmission/turtle/`.

[18] Adida, B.; Herman, I.; Sporny, M. and Birbeck, M. *RDFa 1.1 Primer*. en. other Working Group Note-. World Wide Web Consortium, June 2012.

[19] Bray, T.; Paoli, J.; Sperberg-McQueen, C.M.; Maler, E. and Yergeau, F., ed. *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. W3C Recommendation. Available at `http://www.w3.org/TR/2006/REC-xml-20060816/`. World Wide Web Consortium, Aug. 2006.

[20] Crockford, D. *The application/json Media Type for JavaScript Object Notation (JSON)*. RFC 4627. IETF, July 2006.

[21] Brickley, D., ed. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation. Available at `http://www.w3.org/TR/rdf-schema/`. World Wide Web Consortium, Feb. 2004.

[22] McGuinness, D. and van Harmelen, F., ed. *OWL Web Ontology Language: Overview*. W3C Recommendation. Available at `http://www.w3.org/TR/owl-features/`. World Wide Web Consortium, Feb. 2004.

[23] Motik, B.; Patel-Schneider, P. F.; Parsia, B.; Bock, C.; Fokoue, A.; Haase, P.; Hoekstra, R.; Horrocks, I.; Ruttenberg, A.; Sattler, U. and Smith, M. *OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax*. Last Call Working Draft. (to be published, may be superseded). W3C, 2008. URL: `http://www.w3.org/2007/OWL/draft/owl2-syntax/`.

[24] Boley, H.; Hallmark, G.; Kifer, M.; Paschke, A.; Polleres, A. and Reynolds, D. *RIF Core Design*. World Wide Web Consortium, Working Draft WD-rif-core-20081218. Dec. 2008.

[25] Prud'hommeaux, E. and Seaborne, A., ed. *SPARQL Query Language for RDF*. W3C Recommendation. Available at http://www.w3.org/TR/rdf-sparql-query/. World Wide Web Consortium, Nov. 2007.

[26] Sporny, M.; Inkster, T.; Story, H.; Harbulot, B. and Bachmann-Gmür, R. *WebID 1.0 - Web Identification and Discovery (Draft)*. W3C Specification. W3C, Feb. 2011. URL: http://www.w3.org/2005/Incubator/webid/spec/.

[27] Carroll, J. J.; Bizer, C.; Hayes, P. J. and Stickler, P. „Named graphs". In: *J. Web Sem.* 3.4 (2005), pp. 247–267.

[28] Postel, J. and Reynolds, J. K. *RFC 959: File Transfer Protocol*. Oct. 1985. URL: ftp://ftp.internic.net/rfc/rfc959.txt.

[29] Holtman, K. and Mutz, Andrew H. *Transparent Content Negotiation in HTTP*. Internet RFC 2295. Mar. 1998.

[30] Sequeda, J. *The Semantic Web Has Gone Mainstream! Wanna Bet?* http://semanticweb.com/the-semantic-web-has-gone-mainstream-wanna-bet_b27329. Mar. 13, 2012.

[31] Berners-Lee, T. *Read-Write Linked Data*. http://www.w3.org/DesignIssues/ReadWriteLinkedData.html. Oct. 8, 2011.

[32] Lanthaler, M. and Gutl, C. „Aligning Web Services with the Semantic Web to Create a Global Read-Write Graph of Data". In: *Proceedings of the 2011 IEEE 9th European Conference on Web Services (ECOWS 2011)*. Lugano, Switzerland, Sept. 2011, pp. 15–22.

[33] Bertails, A.; Hawke, S. and Herman, I. *Linked Data Platform (LDP) Working Group Charter*. http://www.w3.org/2012/ldp/charter.html. May 2012.

[34] Gil, Y.; Cheney, J.; Groth, P.; Hartig, O.; Miles, S.; Moreau, L. and Pinheiro da Silva, P. *Provenance Interchange Working Group Charter*. http://www.w3.org/2011/01/prov-wg-charter. Apr. 2011.

[35] Hayes, P. and Franconi, E. *Why must the web be monotonic?* W3C www-rdf-logic mailing list thread, starting at http://lists.w3.org/Archives/Public/www-rdf-logic/2001Jul/0065.html. July 2001.

[81] RLG/OCLC Working Group on Digital Archive Attributes. *Trusted Digital Repositories: Attributes and Responsibilities*. 2002. URL: http://www.oclc.org/programs/ourwork/past/trustedrep/repositories.pdf.

[82] Brand, S. „Escaping The Digital Dark Age". In: *Library Journal* 124, Issue 2 (Mar. 2003), pp. 46–49.

[83] Consultative Committee for Space Data Systems. *Reference Model for an Open Archival Information System (OAIS)*. Available at http://public.ccsds.org/publications/archive/650x0b1.pdf. Jan. 2002.

[84] Messina, A.; Boch, L.; Dimino, G.; Bailer, W.; Schallauer, P.; Allasia, W. and Basili, R. „Creating rich Metadata in the TV Broadcast Archives Environment: the PrestoSpace project". In: *IEEE AXMEDIS06 Conference Second International Conference on Automated Production of Cross Media Content for Multi-Channel Distribution*. 2006, pp. 193–200.

[85] Ferreira, M.; Baptista, A. A., and Ramalho, J. C. „An intelligent decision support system for digital preservation". In: *International Journal on Digital Libraries* 6(4) (2007), pp. 295–304.

[86] Hunter, J. and Choudhury, S. „PANIC: an integrated approach to the preservation of composite digital objects using Semantic Web services". In: *International Journal on Digital Libraries* 6(2) (2006), pp. 174–183.

[87] Gil, Y.; Cheney, J.; Groth, P.; Hartig, O.; Miles, S.; Moreau, L.; da Silva, P. P.; Coppens, S.; Garijo, D.; Gomez, J. M.; Missier, P.; Myers, J.; Sahoo, S. and Zhau, J. *Provenance XG Final Report*. Available at http://www.w3.org/2005/Incubator/prov/XGR-prov/. 2010.

[88] Hartig, O. and Zhao, J. „Publishing and Consuming Provenance Metadata on the Web of Linked Data". In: *Proceedings of the 3rd International Provenance and Annotation Workshop IPAW*. Available at http://olafhartig.de/files/HartigZhao_Provenance_IPAW2010_Preprint.pdf. 2010.

[89] Nilsson, M. and Powell, A. and Johnston, P. and Naeve, A. *Expressing Dublin Core metadata using the Resource Description Framework (RDF)*. Available at http://dublincore.org/documents/dc-rdf/. 2007.

[90] Doerr, M.; Hunter, J. and Lagoze C. „Towards a Core Ontology for Information Integration ". In: *Journal of Digital information* (2003).

[91] Lagoze, C. and Van de Sompel ,H. *The open archives initiative protocol for metadata harvesting - version 2.0*. Available at `http://www.openarchives.org/OAI/openarchivesprotocol.html`. 2002.

[92] Chappell, D. A. *Enterprise Service Bus*. O'Reilly Media, 2004, pp. 1–288.

[93] Box, D.; Ehnebuske, D.; Kakivaya, G.; Mayman, A.; Mendelsohn, N.; Frystyk Nielsen, H.; Thatte, S. and Winer, D. *Simple Object Access Protocol (SOAP) 1.1*. Available at `http://www.w3.org/TR/soap/`. 2000.

[94] Clark, J., ed. *XSL Transformations (XSLT) – Version 1.0*. W3C Recommendation. Available at `http://www.w3.org/TR/xslt`. World Wide Web Consortium, Nov. 1999.

[95] Boyko, A.; Kunze, J.; Littman, J.; Madden, L. and Vargas, B. *The BagIt File Packaging Format (V0.96)*. Available at `https://confluence.ucop.edu/download/attachments/16744580/BagItSpec.pdf?version=1`. 2009.

[96] Van de Sompel, H; Sanderson, R.; Nelson, M.L.; Balakireva, L.; Shankar, H. and Ainsworth, S. „Memento: Time Travel for the Web". In: *CoRR* abs/0911.1112 (2009).

[97] Van de Sompel, H; Sanderson, R.; Nelson, M.L.; Balakireva, L.; Shankar, H. and Ainsworth, S. „An HTTP-Based Versioning Mechanism for Linked Data". In: *CoRR* abs/1003.3661 (2010).

[98] Moreau, L.; Hartig, O.; Simmhan, Y.; Myers, J.; Lebo, T.; Belhajjame, K. and Miles, S. *PROV-AQ: Provenance Access and Query*. Tech. rep. W3C, 2012. URL: `http://www.w3.org/TR/prov-aq/`.

[99] Belhajjame, K.; Cheney, J.; Corsar, D.; Garijo, D.; Soiland-Reyes, S.; Zednik, S. and Zhao, J. *PROV-O: The PROV Ontology*. Tech. rep. W3C, 2012. URL: `http://www.w3.org/TR/prov-o/`.

[100] Bizer, C.;Heath, T. and Berners-Lee, T. „Linked Data – The Story So Far". In: *Int. J. Semantic Web Inf. Syst.* 5.3 (2009), pp. 1–22. URL: `http://eprints.ecs.soton.ac.uk/21285/`.

[101] Auer, S.; Bizer, C.; Kobilarov, G.; Lehmann, J. and Ives, Z. „DBpedia: A Nucleus for a Web of Open Data". In: *In 6th International Semantic Web Conference, Busan, Korea*. Springer, 2007, pp. 11–15.

[102] Tummarello, G.; Delbru, R. and Oren, E. „Sindice.com: Weaving the open linked data". In: *In Proceedings of the International Semantic Web Conference (ISWC)*. 2007.

[103] Jentzsch, A.; Isele, R. and Bizer, C. „Silk - Generating RDF Links while publishing or consuming Linked Data". In: *9th International Semantic Web Conference (ISWC2010)*. Nov. 2010. URL: http://data.semanticweb.org/conference/iswc/2010/paper/519.

[104] Hartig, O.; Bizer, C. and Freytag, J. „Executing SPARQL Queries over the Web of Linked Data". In: *In Proceedings of the 8th International Semantic Web Conference (ISWC)*. 2009. URL: %5Curl%7Bhttp://www2.informatik.hu-berlin.de/~hartig/files/HartigEtAl_QueryTheWeb_ISWC09_Preprint.pdf%7D.

[105] Schwarte, A.; Haase, P.; Hose, K.; Schenkel, R. and Schmidt, M. „FedX: A Federation Layer for Distributed Query Processing on Linked Open Data". In: *ESWC (2)*. Vol. 6643. Lecture Notes in Computer Science. Springer, 2011, pp. 481–486. ISBN: 978-3-642-21033-4.

[106] Quilitz, B. and Leser, U. „Querying Distributed RDF Data Sources with SPARQL". In: *5th European Semantic Web Conference (ESWC2008)*. June 2008, pp. 524–538. URL: http://data.semanticweb.org/conference/eswc/2008/paper/168.

[107] Görlitz, O. and Staab, S. „SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions". In: *COLD*. Ed. by Hartig, O.; Harth, A. and Sequeda, J. Vol. 782. CEUR Workshop Proceedings. CEUR-WS.org, 2011.

[108] Alexander, K.; Cyganiak, R.; Hausenblas, M. and Zhao, J. „Describing Linked Datasets On the Design and Usage of voiD , the Vocabulary Of Interlinked Datasets". In: *Design* 19 (2009). Ed. by Bizer, C.; Heath, T.; Berners-Lee, T. J. and Idehen, K. URL: http://events.linkeddata.org/ldow2009/papers/ldow2009_paper17.pdf.

[109] Harth, A.; Hose, K.; Karnstedt, M.; Polleres, A.; Sattler, K. U. and Umbrich, J. „Data summaries for on-demand queries over linked data". In: *Proceedings of the 19th international conference on World wide web*. WWW '10. Raleigh, North Carolina, USA: ACM, 2010, pp. 411–420. ISBN: 978-1-60558-799-8. DOI: 10.1145/1772690.1772733. URL: http://dx.doi.org/10.1145/1772690.1772733.

[110] Fielding, R. T. and Taylor, R. N. „Principled design of the modern Web architecture". In: *Proceedings of the 22nd international conference on Software engineering*. ICSE '00. Limerick, Ireland: ACM, 2000, pp. 407–416. ISBN: 1-58113-206-9. DOI: 10.1145/337180.337228. URL: http://doi.acm.org/10.1145/337180.337228.

[111] Filho, O. F. F. and Ferreira, M. A. G. V. „Semantic Web Services: A RESTful Approach". In: *IADIS International Conference WWWInternet 2009*. IADIS, 2009, pp. 169–180. URL: http://fullsemanticweb.com/paper/ICWI.pdf.

[112] Martin, D.; Burstein, M.; Hobbs, E.; Lassila, O.; Mcdermott, D.; McIlraith, S.; Narayanan, S.; Parsia, B.; Payne, T.; Sirin, E.; Srinivasan, N. and Sycara, K. *OWL-S: Semantic Markup for Web Services*. Tech. rep. W3C, Nov. 22, 2004. URL: http://www.w3.org/Submission/OWL-S/.

[113] Hadley, M. J. *Web application description language (WADL)*. Tech. rep. Mountain View, CA, USA: Sun Microsystems, Inc., 2006.

[114] Krummenacher, R.; Norton, B. and Marte, A. „Towards Linked Open Services and Processes". In: *FIS*. Ed. by Berre, A.-J.; and Gómez-Pérez, A.; Tutschku, K. and Fensel, D. Vol. 6369. Lecture Notes in Computer Science. Springer, 2010, pp. 68–77. ISBN: 978-3-642-15876-6.

[115] Speiser, S. and Harth, A. „Integrating Linked Data and Services with Linked Data Services". In: *ESWC (1)*. Vol. 6643. Lecture Notes in Computer Science. Springer, 2011, pp. 170–184. ISBN: 978-3-642-21033-4.

[116] Verborgh, R.; Steiner, T.; Van Deursen, D.; De Roo, J.; Van de Walle, R. and Gabarró Vallés, J. „Capturing the functionality of Web services with functional descriptions". In: *Multimedia Tools and Applications* 2 (2013), pp. 365–387. DOI: 10.1007/

s11042-012-1004-5. URL: http://link.springer.com/article/10.1007/s11042-012-1004-5.

[117] Prud'hommeaux, E., ed. *SPARQL 1.1 Query Language*. W3C Recommendation. Available at http://www.w3.org/TR/sparql11-query/. World Wide Web Consortium, Nov. 2012.

[118] Glimm, B. and Ogbuji, C., ed. *SPARQL 1.1 Entailment Regimes*. W3C Recommendation. Available at http://www.w3.org/TR/sparql11-entailment/. World Wide Web Consortium, Jan. 2013.

[119] Anicic, D.; Fodor, P.; Rudolph, S. and Stojanovic, N. „EP-SPARQL: a unified language for event processing and stream reasoning". In: *Proceedings of the 20th international conference on World wide web*. WWW '11. Hyderabad, India: ACM, 2011, pp. 635–644. ISBN: 978-1-4503-0632-4. DOI: 10.1145/1963405.1963495. URL: http://doi.acm.org/10.1145/1963405.1963495.

[120] Ying, L. and Shujuan, J. „International Workshop on LarKC - a Platform for Massive Distributed Incomplete Reasoning". In: *Digital Library Forum, China Science and Technology Information Institute, Beijing* (Apr. 2011).

[121] Serafini, L. and Tamilin, A. „Drago: Distributed reasoning architecture for the semantic web". In: *ESWC*. Springer, 2005, pp. 361–376.

[122] De Roo, J. „EYE reasoner". In: URL: http://eulersharp.sourceforge.net/.

[123] Sirin, E.; Parsia, B.; Cuenca Grau, B.; Kalyanpur, A. and Katz, Y. „Pellet: A Practical OWL-DL Reasoner". In: *Web Semantics* 5.2 (2007), pp. 51–53. ISSN: 1570-8268. DOI: 10.1016/j.websem.2007.03.004.

[124] Tsarkov, D. and Horrocks, I. „FaCT++ description logic reasoner: System description". In: *In Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006*. Springer, 2006, pp. 292–297.

[125] Haarslev, V.; Hidde, K.; Möller, R. and Wessel, M. „The RacerPro knowledge representation and reasoning system". In: *Semantic Web* 3.3 (2012), pp. 267–277.

[126] Glimm, B.; Horrocks, I.; Motik, B.; Shearer, R. and Stoilos, G. „A novel approach to ontology classification". In: *J. Web Sem.* 14 (2012), pp. 84–101.

[127] Polleres, A.;Scharffe, F. and Schindlauer, R. „SPARQL++ for Mapping Between RDF Vocabularies". In: *OTM Conferences (1)*. Ed. by Meersman, R. and Tari, Z. Vol. 4803. Lecture Notes in Computer Science. Springer, 2007, pp. 878–896. ISBN: 978-3-540-76846-3.

My warm thanks to all the research partners
who participated in this study.