Ontology Theory, Management and Design:
Advanced Tools and Models

Faiez Gargouri
Higher Institute of Informatics and Multimedia of Sfax, Tunisia

Wassim Jaziri
Higher Institute of Informatics and Multimedia of Sfax, Tunisia

Information Science REFERENCE

INFORMATION SCIENCE REFERENCE

Hershey · New York

Director of Editorial Content: Kristin Klinger
Director of Book Publications: Julia Mosemann
Acquisitions Editor: Lindsay Johnston
Development Editor: Joel Gamon
Publishing Assistant: Sean Woznicki
Typesetter: Myla Harty
Production Editor: Jamie Snavely
Cover Design: Lisa Tosheff
Printed at: Yurchak Printing Inc.

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com/reference

Library of Congress Cataloging-in-Publication Data

Ontology theory, management, and design : advanced tools and models / Faiez Gargouri and Wassim Jaziri, editors.
p. cm.
Includes bibliographical references and index.
Summary: "The focus of this book is on information and communication sciences, computer science, and artificial intelligence and provides readers with access to the latest knowledge related to design, modeling and implementation of ontologies"--Provided by publisher. ISBN 978-1-61520-859-3 (hardcover) -- ISBN 978-1-61520-860-9 (ebook) 1. Ontologies (Information retrieval) 2. Knowledge representation (Information theory) 3. Artificial intelligence. I. Gargouri, Faiez, 1965- II. Jaziri, Wassim, 1975-
TK5105.88815.O588 2010
004--dc22
2009053383

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 4
An Algebra of Ontology Properties for Service Discovery and Composition in Semantic Web

Yann Pollet
CEDRIC Laboratory, France

ABSTRACT

The authors address in this chapter the problem of the automated discovery and composition of Web Services. Now, Service-oriented computing is emerging as a new and promising paradigm. However, selection and composition of Services to achieve an expected goal remain purely manual and time consuming tasks. Basing our approach on domain concept definitions thanks to an Ontology, the authors develop here an algebraic approach that enables to express formal definitions of Web Service semantics as well as user information needs. Both are captured by the means of algebraic expressions of ontology properties. They present an algorithm that generates efficient orchestration plans, with characteristics of optimality regarding Quality of Service. The approach has been validated by a prototype and an evaluation in the case of an Health Information System.

INTRODUCTION

The number of available Web data sources and services has exploded during the last years. This enables users to access rich information in many domains such as health, life sciences, law, geography, and many other domain of interest. Thanks to this wealth, users rely more on various digital tasks such as data retrieval from both public and corporate data sources and data analysis with Web tools or services organized in complex workflows [Gao, 2005, Kinsi,2007]. However, human users have to spend uncountable hours to explore and discover Web resources that meet their requirements. In addition, in many cases, users need to compose a specific set of Web resources in order to fulfill a complex question. This situation is mainly due to the inability of present standards in capturing Web Service semantics, i.e. the precise meaning of what a given Web Service exactly delivers regarding a specific user context.

DOI: 10.4018/978-1-61520-859-3.ch004

Meanwhile, Service-oriented computing (SoC) is emerging as a new and promising computing paradigm that centers on the notion of service as the fundamental element for accessing heterogeneous, rich and distributed resources in an interoperable way [Roman, 2005]. Web services are self-describing components that support a rapid and significant reuse of distributed applications. They are offered by service providers, which procure service implementation and maintenance, and supply service descriptions. Service descriptions are used to advertise service capabilities, behavior, Quality of Service, etc. (UDDI, WSDL, OWL-S). Service descriptions are meant to be used by other applications (and possibly other services), and not only by humans. WSDL and UDDI are the basic standards used for Web Service capabilities descriptions and advertising. However, they focus on the description of interfaces and syntactic considerations.

So, at present, the development of powerful applications on the Web is still facing two major problems. The first one is related to the increasing difficulties of identifying services that perform a specific task. The second one concerns the difficulty to orchestrate and compose services in a smooth, automated, and, if possible, optimal way, regarding the Quality of Service (QoS). This is still very challenging for many reasons. The main raison is the present limited ability of languages and models to describe the semantic of Web Services, despite tremendous efforts driven by the semantic Web Services community [Roman 2005, Kopecki 2007, Martin 2007].

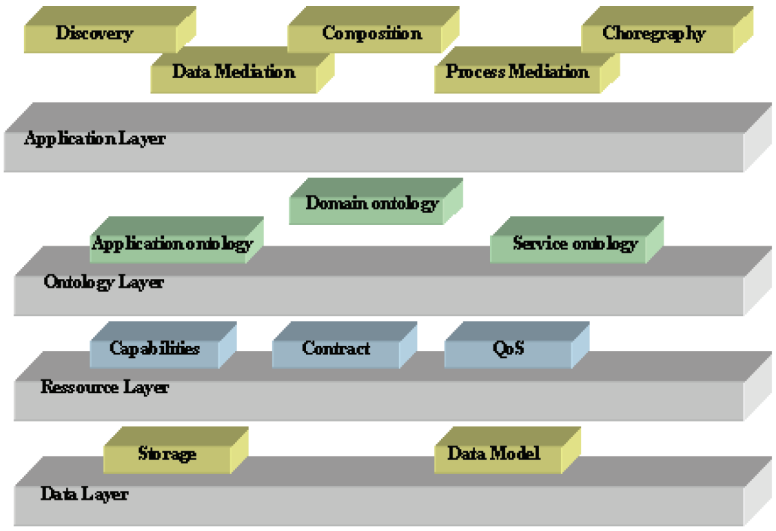
In order to increase the benefits gained from rich Web resources, it would be of the highest importance to express formal semantic descriptions of Web Services. Such descriptions are in fact the absolute requisite condition to enable assisted or automated selection of relevant Web Services, and generate meaningful compositions of them. In addition, *non functional aspects* such as QoS (performance, availability, …) should be taken into account at Services selection and for the generation of a composition plan. This remains at present challenging and hard issue.

BACKGROUND

Emerging infrastructures such as the Semantic Web [Berners-Lee, 2001], the Semantic Grid [Goble, 2005] and Service Oriented architectures [Roman, 2005], support on-line access to a large number of resources from data sources and Web services to knowledge representation models such as taxonomies and ontologies. Ontologies play an important role in the Semantic Web and provide the basics for the definition of concepts and relationships that make information integration possible. OWL-S is proposed as a way to express more detailed descriptions of Web Services via a provided ontology of Web Services. But it remains limited and fails in expressing what a Service really provides, although services should ideally export also their semantics.

The new Semantic Service-Oriented Architecture (SSOA) leverages rich, machine-interpretable descriptions of data, services and processes to enable software agents to automatically interact and achieve collaborative goals. The SSOA integrates the principles of Service-Oriented Computing with semantics-based computing, Typically, a Semantic Service-Oriented Architecture (SSOA) includes four layers: the *data* layer, the *resource* layer, the *ontology* layer, and the *community* layer, as depicted in figure 1. The *data* layer represents data published by Web resources, and the hyperlinks that interconnect theses data objects, for example PubMed publications or medical records stored in Google Health. The *resource* layer is comprised of Web resources and their links, Resources can be either data source, e.g., SwissProt which is a protein database, or a Web service, e.g., BLAST which is a bioinformatics Web-based alignment tool. In the case of a data source, a resource implements some concepts and individuals of the ontology level, while in the case of Web Services, they

Figure 1. A layered web architecture



implement a semantic link that relates input and outputs parameters. Web Services infrastructure provides the syntactical basis for interoperability between resources thanks to standards such as WESL [Akkiraju, 2005], UDDI, and SOAP.

Semantic Web service (SWS) technology aims at providing richer semantic specifications of Web services in order to enable the flexible automation of service processes. The field includes substantial bodies of work to enhance resource descriptions with the use of an ontology including OWL for Services (OWL-S) [Martin, 2007], the Web Service Modeling Ontology (WSMO) [Roman, 2005], and SAWSDL [Kopecki, 2007]. Some approaches, such as [Ayadi, 2008] introduce a canonical set of semantic descriptions of Web services in order to extend SAWSDL standard and support automatic reasoning.

Regarding Service discovery, several techniques have been proposed to support service discovery using logical inference. Existing solutions, including those of Paolucci et al. [Paolucci, 2002, 2007] and Sycara et al. [Sycara, 2003, 2006] propose a method based on DAML-S descriptions for matching goals and capabilities of semantic Web services. Sycara et al. describe the implementation of the DAML-S/UDDI matchmaker that expands UDDI by providing semantic capability matching. OWLS-MX [Klusch, 2006] is a hybrid matchmaker that complements logic based reasoning with approximate matching based on similarity computations.

The proposed FUSION semantic registry [Kourtesis, 2008] relies on a combination of three standards: UDDI, for storing and retrieving syntactic and semantic information about services and service providers, SAWSDL for creating semantically annotated descriptions of service interfaces, and OWL, for modeling service characteristics and performing fine-grained service matchmaking via Description Logic reasoning. In contrast with prominent approaches, FUSION relies on functional and *non-functional* properties for matchmaking. However, it is not clear in [Kourtesis, 2008] how service discovery is realized in this System.

There is today a wide agreement on the fact that a flexible Web Service infrastructure, where resources can be discovered and smoothly composed into workflows, is strongly required. But,

in spite of tremendous efforts around semantic Web Services, the automation of these tasks is still challenging and hard to achieve.

AN ALGEBRAIC APPROACH FOR SERVICE DISCOVERY AND COMPOSITION

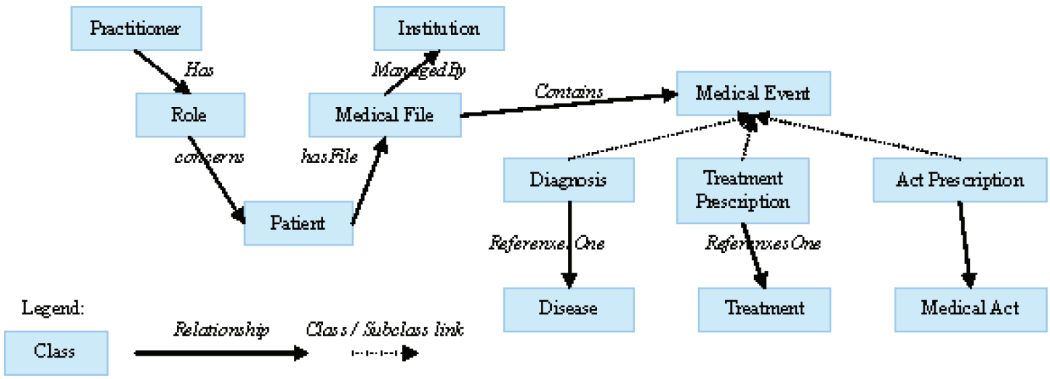
We explore here an algebraic approach for Service Discovery and composition based on an ontology of a given application domain. In order to motivate the need for an automated selection and execution of relevant Services, we present at first a simplified case study in healthcare domain. Then we present the RCS algebra (Relationship Composition with Structured Expressions) that we propose as a new theoretical basis, and detail its mathematical foundations. We present then a mapping model based on the Algebra that enables to formally express the semantics of Services, referring to domain ontology. We present also an efficient algorithm for execution plans generation developed on the basis of our approach. At last, we give a short presentation of a developed software framework and of its application to the case study in order to validate our approach.

A Motivating Example

As a way to illustrate our problem, we present here a case study issued from the Healthcare domain. In a given regional area, we have several healthcare institutions (hospital and practitioner offices), each of them managing data about their patients. A medical file is a set of time labeled *medical events* from different types (diagnosis, treatment prescription, medical act), with standard codifications. An institution may have zero, one or several medical files for a given patient. Each practitioner from an institution has access rights regarding a patient file, depending of his /her role in the healthcare process of this patient (referent practitioner, consultant, etc.). Access rights may be limited in time, e.g. for some roles in relationship with specific acts. In addition to medical files, it may exist Identities Servers, at regional level, that deliver information about patient's administrative details, as well as links to existing medical files on the basis of a patient identity. The figure 2 below illustrates a fragment of a relevant simplified ontology (based on a consistent combination of different existing standards). For a concern of readability, datatype properties (attributes) and cardinalities don't appear.

A practitioner has access to a secured infrastructure where the different information servers that deliver data about patients are accessed via Web Services.

Figure 2. A simplified fragment of ontology in medical practice domain



In healthcare domain, therapeutic decision, e.g. decision in oncology, requires access to various pieces of information scattered among various several institution servers. Therapeutic decision is the most convincing use case regarding the requirement for Services selection and composition, as there are many sources from which data have to be retrieved, with possible alternatives. But there exist many examples of support applications leading to the same requirements, such as epidemiologic studies, and access to anonymous patient files for medical students. Plans should be flexible in order to be automatically adapted when new sources are added to the community.

Problem and Basic Hypothesis

We assume here that the various information sources provide access to relevant data by the means of *Services*. We define here a *Service* just as a black box function that may be invoked by a distant software entity, with input data, and delivering output results. We make no assumption about technology, and these *Services* may be implemented as Web Services, or thanks to another technology. We consider here *stateless data access Services*, excluding *Services* having a side effect on internal data and/or on external world. A *Service* encapsulates all the details of operations executed to deliver a required piece of information. In particular, the user entity doesn't know whether the output is extracted from a local database, results from a calculation, or from a combination of both (e.g. rights determination from roles in our example).

We consider a domain Ontology O. This Ontology defines a set of classes $\{C_i\}$, each of these having some *attributes* proprieties $\{V_{i,j}\}$ (*datatype properties*), and *directed relationships* $\{R_{i,k}\}$ to other classes (*object properties*).

By hypothesis, a *Service* will be such as $\{x'\} = S(x_1, \ldots, x_n)$, or $\{v\} = S(x_1, \ldots, x_n)$, where x' et x_i are *individuals* whose types directly correspond to Ontology classes, and where v is a type corresponding to a *datatype* in O. In the following, we shall also consider *Services* with more than a single output parameter. The development of relevant wrapping code, in the case of *Service* reuse, is out of the scope of the issue addressed by the chapter.

Principles

The domain Ontology provides a well defined formalisation of the various concepts of the domain, with meaningful properties and relationships. This enables to attach a precise meaning to a given piece of information, in particular when required by a user. However, the concern of a user may not exactly correspond to an Ontology concept. For example, the exams and the treatments that have been provided to a patient P have an interest for some users; nevertheless, this concept does not immediately correspond to a property of patient class. To define such a result, we have to consider at first all the medical files associated to the given individual patient, the union of all medical events from each file, and then the extraction of exams and treatments. There are in fact two problems. The first one is that we want to deal with an indirect access to medical events from a patient, with restriction condition on medical events. The second one is the fact that the type of results does not match with a concept known in the Ontology.

In order to define such information, we shall introduce the notion of *derived* property. A *derived* property will be formally defined by the means of an algebraic expression of the Ontology properties. A *derived* property will be attached to an Ontology class, or to new class defined on the basis of existing Ontology classes, and called here a *derived* class. A *native* or *derived* property will be called an *extended* property. So, a piece of information needed by a user will be always defined by both an individual, and an *extended* property to evaluate. This is the first principle of our approach.

The second principle concerns the capture of *Services* semantics. It consists in defining the semantic of a *Service*, i.e. the link between input and output by a correspondence with a property of the Ontology, such a *relationship* or an *attribute*. The most simple case is this where a *Service*, with an individual x as input, delivers as output the set of objects {x'} in correspondence with x via a given propriety R. As an example, if a *Service* delivers the content of a medical file starting from the reference to this file as input, the meaning of the *Service* is perfectly defined by the relationship "Contains" of the Ontology. Our principle is therefore this of defining the semantic of a *Service* by an Ontology property that this *Service* may *realize*. We shall call such a correspondence a *mapping* between a *Service* and a property. However, this is only a specific case, and there is no reason for a given *Service* to *realize* exactly a particular Ontology relationship. First of all, a *Service* may directly *realize* an indirect correspondence. For example this is the case if a *Service* delivers the overall content of a patient file given the patient's identity. This is the case in which a *Service realizes* a *derived* property. So, a *Service* may realize a *native* or a *derived* property.

Another case is this of a *partial realization*. Consider a *Service* provided by an institution delivering a set of medical events, for a given patient. The *Service* will be able, of course, to deliver results only for patients known in the institution, i.e., that have at least one event in this institution. In addition, it will be able to deliver only known events, i.e. those which have been performed in this institution. We shall present a general *mapping* model that enables to express sophisticated semantic correspondences between an available *Service* and properties. This model covers the more complex cases involving *Services* with more than one input parameter and/or more than one output parameter. The figure 3 synthesizes the two principles of our approach.

The RCS Algebra

We present here the RCS algebra that enables to express formal definition of new properties.

Derived Classes

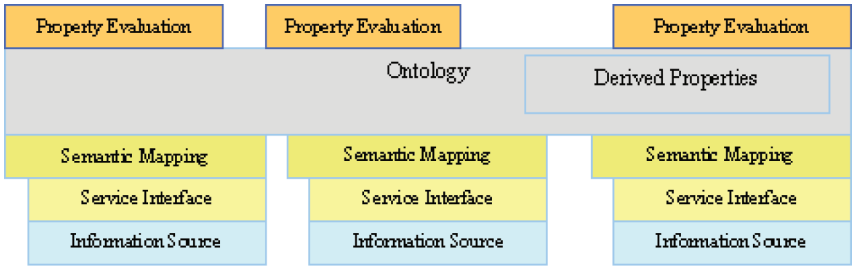
Starting from the classes defined in the Ontology, called here *natives* classes, one can define new classes by application of the following OWL operators and their combinations. Such new classes will be called *derived* classes.

Intersection: $C = C_1.C_2$ ($C_1 \cap C_2$) defined by: $x \in C_1.C_2$ iff $x \in C_1$ AND $x \in C_2$, where C_1 and C_2 are *natives* classes, or already defined *derived* classes. Properties which are valid for $x \in C$ are those valid for $x \in C_1$ or for $x \in C_2$

Union: $C = C_1 + C_2$ ($C_1 \cup C_2$) defined by: $x \in C_1 + C_2$ iff $x \in C_1$ OR $x \in C_2$, where C_1 and C_2 are *natives* or already defined *derived* classes, Valid properties are those valid for $x \in C_1$ and $x \in C_2$

Property Restriction: $C' = C^P$ is defined by $x \in C^P$ iff $x \in C$ AND P(x), where C is a *native* or an already defined *derived* class, and P a predi-

Figure 3. Organization of the ontology based framework



cate defined as a logical expression AND-OR of elementary predicates of the form $V_i \Theta v$, where V_i is a property defined on C, v a value of this property, and where Θ a comparison operator defined on the value domain. Valid properties for an $x \in C^p$ are those valid for $x \in C$

Property (Ontology Lattice). Let be an Ontology O, the set of native classes of O completed by the set of derived classes generated by intersection, union, and property restriction is a Lattice. The associated pseudo order is « ≤ », defined by: $C_1 \leq C_2$ iff C_1 c C_2 (set inclusion) and IsProperty (C_2) → IsProperty (C_1). This extends the Ontology relationship of specializations / generalization. A native or derived class will be called an extended class.

Algebraic Operators on Properties

We introduce here two binary operators on relationships: the composition and the union, plus a third operator called relationship restriction. These operators apply on *extended* (i.e. *native* of *derived*) relationships. In addition to relationships operators, we introduce the projection that enables to deal with attribute properties. We shall write <R, x, y> to express that the individual x and y are in relationship by R. In addition, dom(R) and range(R) will respectively denote the domain of a relationship R (i.e., the set of x), and its range (i.e. the set of y). At last, minCard(R) and maxCard(R) will respectively denote the minimum and maximum cardinalities of R (by default minCard(R) = 0 and maxCard(R) = ∞).

Composition

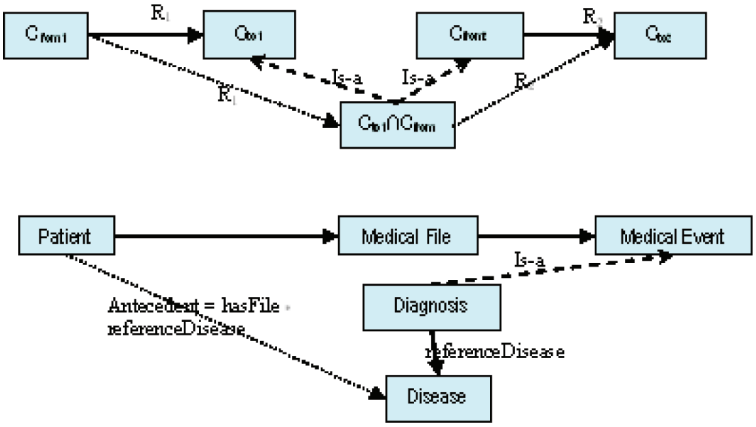
Definition (composition operator). Given two relationships R_1 and R_2, the composition $R_1 * R_2$ is the relationship such as $dom(R_1 * R_2) = dom(R_1)$, $range(R_1 * R_2) = range(R_2)$, and <R, x, z> iff there exists $y \in range(R_1) \cap dom(R_2)$ such as $<R_1, x, y>$ and $<R_2, y, z>$

The result is defined for any R_1 and R_2 and belongs to the set of relationships R. * is *associative* but not *commutative*. If $range(R_1) \cap dom(R_2) = \emptyset$, then R is a *null* relationship, with no value for any individual. In addition, we have $minCard(R_1 * R_2) = minCard(R_1) \cdot minCard(R_2)$, and $maxCard(R_1 * R_2) = maxCard(R_1).maxCard(R_2)$. The figure below illustrates the principle of the composition operator and an example of composition:

Union

Definition (union operator). Given two relationships R_1 and R_2, the union $R = R_1 + R_2$ is the relationship such as $dom(R_1 + R_2) = dom(R_1) \cap$

Figure 4. Illustration of the composition operator



dom(R_2), range (R_1 + R_2) = range (R_1) U range (R_2), and <R, x, y> iff <R_1 x, y> or <R_2, x, y>

The set of individuals {y} associated to an individual x by R is so the union of the two sets of individuals respectively associated to x by R_1 and by R_2. R_1 + R_2 is always defined as a relationship, possibly empty (in particular in the case where dom(R_1) ∩ dom(R_2) = Ø). The union operator is *commutative* and *associative*. The composition operator $_*$ is *distributive* with respect to the union +. We have: minCard(R_1 + R_2) = Max (minCard(R_1), minCard(R_2)) and maxCard(R_1 + R_2) = Min (maxCard(R_1), maxCard(R_2))

Restriction Filter and Relationship Restriction

Definition (restriction filter). Given two classes C_1 and C_2, the relationship [C_1, C_2], called restriction filter from C_1 to C_2, is the relationship such as dom([C_1, C_2]) = C_1, range([C_1, C_2]) = C_2, and < [C_1, C_2], x, y> iff (x=y) and (x ϵ C_1 ∩ C_2)

Such a relationship is a *constant* as it is canonically defined from any ordered pair of classes, independently of the domain ontology relationships. [C_1, C_2] associates to any individual from C_1 either the individual itself, either an empty value set. An individual y from C_2 may be associated to an x from C_1 by [C_1, C_2] only if y ϵ C_1. If C_1 = C_2 = Universal, so [C_1, C_2] is the *Identity* relationship. If C_2 = C_1^P, where P is a predicate defined on C_1, [C_1, C_1^P] is a classical P-predicate restrictor. In particular, we have the following property:

$$[C, C^{PRE1}] _* [C, C^{PRE2}] = [C, C^{PRE1 \text{ AND } PRE2}]$$

So, we can define the *relationship restriction* operator in the following way:

Definition (restriction). Given a relationship R, two predicates PRE(x) and POST(y) respectively applying on individuals x from dom(R) and y from range (R), the relationship restriction of R by PRE and POST is defined by $R^{PRE, POST}$ = [C_1, C_1^{PRE}] $_*$ R $_*$ [C_2^{PRE}, C_2], where C_1 = dom(R) and C_2 = range(R)

So, the relationship $R^{PRE, POST}$ associates to any individual x from C_1 such as PRE its correspondent individual y by R if POST(y) is satisfied, and an empty set if not. The above figure 5 illustrates the relationship restriction operator:

Examples: One may derive from the relationship « contains » new relationships with the same domain (Medical File) and same range (Medical Event), but with a more specific meaning, e.g.:

- **R_1**, that associates to the medical files of a given institution H their content (with an empty value set for other ones): R_1 = Contains $^{ManagedByH, True}$, where ManagedByH (d) = d.managedBy = H
- **R_2**, that associates to any medical files the sets of its diagnosis: R_2 = Contains $^{True, Is(Diagnosis)}$

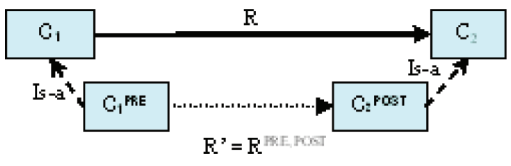
Canonical Decomposition of a Relationship

Let's consider a relationship R, two sets of predicates {PRE_i ; i=1, ..,n} and {$POST_j$; j=1, ..,m} respectively applying on the classes dom(R) and range(R), and such as:

PRE_1 OR …. OR PRE_n = True, and $POST_1$ OR …. OR $POST_m$ = True,

We have: $R = \sum_{i=1,..,n \text{ OR } PREi = True} R^{PREi, True} = \sum_{j=1, \ldots,m \text{ OR } POSTj = True} R^{True, POSTj} = \sum_{i=1,..,n, j=1, \ldots,m \text{ OR } PREi = True, \text{ OR } POSTj = True} R^{PREi, POSTj}$

Figure 5. Illustration of the relationship restriction operator



If R is a *native* relationship, and if the predicate sets PRE_i and $POST_j$ are composed of mutually exclusives conditions, the above property defines a *canonical way to decompose a relationship*.

Algebraic Expressions and Extended Relationships

The above operators enable to formulate algebraic expressions, where operands are *extended* relationships, i.e. *native* Ontology relationships, and/or already defined *derived* ones. Formulas define new *derived* relationships. Their domain and range are defined using the operator rules.

As an example, $R = R_1 + R_2 *_{PRE1, POST1} R_3 +_{PRE2, POST2} R_4 *_{} R_5$ is a relationship with domain $dom(R) = dom(R_1).dom(R_2).dom(R_4)$ and $range(R) = dom(R_1) + dom(R_3) + dom(R_5)$

$R = has^{True,\ role=Referent}_{*}$ concerns $*$ $hasFile^{True,\ managedBy=H}_{*}$ contains $^{True,\ Is(Diagnostic)}$ is a property of domain « Practitioner », and its range is the class « Diagnosis ».

It is possible to transform a given formula into equivalent expressions using the properties of operators (associativity of composition and union, commutativity of union, distributivity). If an operand is an *extended* relationship, another formula should have previously defined it, and we exclude in the present version of the theory cyclic definitions. The set of *derived* relationships associated to the Ontology O will be defined by a sequence of expressions of the form: $R_i = EXPR_i (R_{1,i}, …., R_{ni,\ i})$; 1=1, …, N, where $EXPR_i$ is an expression whose operands $R_{k,\ i}$ $k = 1, …, n_i$ are either *native* relationships, either *derived* relationship taken among the $R_1, …, R_{i-1}$

Example 1: The relationship R_1 that associates to any patient registered in the institution H the set of his/her medical files is: $R_1 = hasFile\ *$ [MedicalFile, MedicalFile $^{managedBy=H}$] $*$ contains.

Example 2: The relationship R_2 that associates to any patient having hepatitis as diagnosis, the institution where his/her has a medical file is:

$R_2 = asFile\ *$ [MedicalFile, MedicalFile $^{Hepatitis\ IN\ medicalFile.contains}$] $*$ managedBy

R_1 and R_2 are *extended* properties of the class "Patient"

Extended Attributes

In order to support the definition of *derived* attribute properties, we define at first the operator of *projection* that gives access to the values of an attribute of a given individual.

Definition (projection). Given a native class C having an attribute property V. The projection C.V of the class C on the attribute V is the function that associates to any individual x from C the set $\{V_i\}$ of the values attached to x by the attribute V.

The projection allow to formulate expressions such as: $V' = EXPR (R_1, …, R_n).V$, where $EXPR (R_1, …, R_n)$ is an algebraic expression of the relationships $R_1, …, R_n$, and where V is an attribute property attached to the class $C' = range (EXPR (R_1, …, R_n))$.

This expression defines a new property attribute attached to the class $dom (EXPR (R_1, …, R_n))$. It is such as $dom(V') = dom (EXPR (R_1, …, R_n))$, and $range(V') = range(V)$. If $V = R.V$, we have $maxCard(V') = maxCard(R).maxCard(V)$ and $minCard(V') = minCard(R).minCard(V)$

Example: (hasFile $*$ contains $*$ [MedicalEvent, Diagnosis]).longName is the property attached to a patient giving his/her various diagnosis names in clear. Such a property will be said *derived attribute*. A *native* or *derived* attribute is said to be an *extended* attribute.

Semantic Services Mapping

Basic Principles

So far, we have at our disposal an Algebra enabling to manipulate and combine the properties of an Ontology in order to *define new properties with*

the help of rigorously defined operators. In this section, we exploit this Algebra for the purpose of *Services* semantic definition. We define here a *mapping model* enabling to express the meaning of a given *Service* (i.e. the meaning of the transformation it performs from its inputs to its outputs) thanks to Ontology properties. We have indicated that the basic principle was this of capturing *Service* semantics by the means of relationships. In the simplest case, the operation performed by a 1-1 *Service* (i.e. a *Service* with one input and one output parameter) may exactly correspond to an Ontology relationship. In the case when there exists a *Service* with a patient identification as input and delivering as output the set of the references to his/her various medical files, the semantic of this *Service* corresponds exactly to the ontology relationship « hasFile». We call here *mapping*, such a semantic correspondence. We shall say that the Service *realizes* the relationship.

For a given practitioner, a *Service* may deliver the set of his/her patients (i.e. for who his/her is referent practitioner), although this relationship does not exist in O. In order to express such a link, we shall considerer more general *mappings*, linking a *Service* to an *extended* relationship. Such a correspondence may be *total*, or *partial* (case where a *Service* realize only a part of the relationship).

However, 1-1 Services are a particular case of a more general n-p *Services*, with n input parameters and p output parameter. So we need a *mapping* model more general that direct association. For that, we introduce *mapping* correspondences linking an output individual to n input individuals by the mean of algebraic expressions.

At last, if the notion of *mapping* expresses what the *Service* delivers regarding meaning, we should also be able to capture *non functional aspects*, such as e.g. performance or availability of Services. We introduce for that a model of *Quality of Service* (QoS) in order to have a complete definition of a *Service*.

Simple Mapping Assertions

We consider here a 1-1 *Service*, whose input parameter is an individual from a class C_{in} = In(S), and output parameter a set of individuals from class C_{out} = Out(S). The output set of S is so $2^{Out(S)} = 2^{Cout}$, i.e.: S: $C_{in} \rightarrow 2^{Cout}$

Definition (realization of a relationship by a service). Given a 1-1 Service, where In(S) and Out(S) are natives or derived classes, and R a native or derived relationship. We shall say that the Service realizes the relationship R iff $y \in S(x) \Leftrightarrow <R, x, y>$

We have so: In(S) = dom(R) and Out (S) = range (R). We shall write <MAP, S, R>. This expression being called a *mapping assertion*.

A service may also *realize* an attribute. Let's consider a 1-1 *Service* S, whose input set In(S) is a class C_{in}, and whose output set Out(S) is 2^D, where D is a datatype such as Interger, String, Date, etc.

Definition (realization of an attribute by a service). Given a 1-1 Service where In(S) is an extended class, and Out(S) a datatype D. Let be V a native or extended attribute of type D. S realizes V iff $v \in S(x) \Leftrightarrow <V, x, v>$. We write: <MAP, S, V>

We can therefore consider *individual-oriented Services*, which deliver sets of individuals from a given class, and *datatype-oriented Services*, which deliver as output sets of datatype values from a given Ontology datatype. In order to simplify the presentation, we describe here the mapping model with only *individual-oriented Services*, only a few extensions being necessary to integrate *datatype-oriented Services*.

Restricted Mappings

A existing *Service* may realize only a part of a relationship R, i.e. only apply to a sub domain of R, and only deliver a sub part of expected results regarding R. There are many reasons for that, the main one being that it is natural that an organiza-

tion only delivers information in its perimeter of influence or knowledge. In our example, it is illustrated on one hand by the relationship "hasFile", and on the other hand by the *Services* provided by the various institutions, that have a view limited to their own patients and events. Such *Services* only realize parts of the "hasFile" relationship, with restrictions on inputs and output individuals.

In this case, we have so a weaker property, i.e.: $\{y_i\} = S(x) \rightarrow <R, x, y_i>$, although the inverse may be false. S is said a *partial realization* of R. We consider here the case where the limitations in R realization follow criteria of rationality, in relationship with some known organizational rules. So, well defined criteria of limitation may be expressed (there exist cases where it is not true, e.g. in the case of a asynchronously lazy replicated databases, where limitation may depend on delay. This case will be captured in our approach by the means of QoS).

So, we consider a more general *mapping* model, based on a new form of *mapping assertions*.

Definition (general mapping assertion). Given a Service S, a relationship R, two predicates PRE and POST respectively applying on dom(R) and range(R), a mapping assertion states that S is a realization $R^{PRE, POST}$, i.e. $(y \in Y = S(x)) \Leftrightarrow (<R, x, y>$ AND PRE(x) AND POST(y_i)). We write: <MAP, S, R, PRE, POST>, that is equivalent to $<MAP, S, R^{PRE, POST}>$

It has to be noticed that this notion of *post condition* on output is not at all this of *effect*, related to the semantic capture of Web Service with side effect as it may be found in the literature.

The Algebra of Services

Let's consider a set of 1-1 *Services* $\{S_i\}$, a set of relationships $\{R_j\}$, and a set of *mapping assertions* $<MAP, S_i, R_j, PRE_{i,j}, POST_{i,j}>$. The relationship $R_j^{PREi,j, POSTi,j}$ defines the semantic of S_i. We define the operators of *Composition*, *Union* and *Restriction* applying on Services. These operators are symmetrical to those applying on relationships:

Definition (composition, union, and restrictions of services). Given three Services S_1, S_2, S, and two predicates PRE and POST respectively applying on In(S) and Out(S).

- $S_1 * S_2$ is defined by: x'' $\in (S_1 * S_2)(x)$ iff there exists x' such as x' $\in S_1(x)$ AND x'' $\in S_2(x')$
- $S_1 + S_2$ is defined by: x' $\in (S_1 + S_2)(x)$ iff x' $\in S_1(x)$ OR x' $\in S_2(x)$
- and $S^{PRE, POST}$ is defined by: x' $\in S^{PRE, POST}(x)$ iff x' $\in$ S (x) AND PRE (x) AND POST (x')

We have: $in(S_1 * S_2) = in(S_1)$, $out(S_1 * S_2) = out(S_2)$, $in(S_1 + S_2) = in(S_1).in(S_2)$, and $out(S_1 + S_2) = out(S_1) + out(S_2)$. The invocation of $(S_{1*}S_2)(x)$ leads to an invocation of S_1, and a number of invocations of S_2 depending of result $\{S_2 (o)\}$ cardinality, that may be 0, 1 or many. In addition to the Services provided by infrastructure, we consider *filters* $F_{[C1, c2]}$, that are predefined Services *realizing* $[C_1, C_2]$ relationships.

An algebraic expression of *Services* is equivalent to an execution graph, where elementary instructions are *Services* invocations, controlled by the means of *composition*, *union* and *restriction* operators that respectively stand for sequence, parallel activation (fork and join) and test conditions. The *Services* provided by the infrastructure are called *real Services*, as *Services* defined by algebraic expressions of *real* Services are *abstract Services*. As an example, consider three *Services* S_1, S_2 and S_3 that respectively realize the relationship « hasFile », the relationship "contains" for public institutions, and the same relationship « contains » other institutions. The complete medical file of a patient P is given by invocation of the abstract Service $S_1 * (S_2 + S_3)$.

If <MAP, S_1, R_1> AND <MAP, S_2, R_2>, we have <MAP, $S_{1*}S_2$, $R_{1*}R_2$> and <MAP, $S_1 + S_2$, $R_1 + R_2$>. If <MAP, S, R>, then we have $<MAP, S^{PRE, POST}, R^{PRE, POST}>$. This defines an algebraic *homomorphism* between the relationship Algebra

and the 1-1 Services Algebra. An important result concerns the evaluation of *derived* relationships with several *Services* realizing *partial* mappings, thank to the following property:

Let be a relationship R, a set of Services S_i, i =1, …, n, such as <MAP, S_i, R, PRE_i, $POST_i$>. The Service $S = \sum_{i=1,\ldots,n} S_i$ realizes the relationship R, i.e. <MAP, $\sum_{i=1,\ldots,n} S_i$, R> iff $OR_{PREi,\ POSTi}$ (PRE_i AND $POST_i$) = True

Similar definitions and results may be developed for datatype oriented *Services*.

Complex Mappings

We consider here n-p *Services*, i.e. with n input parameters and p output parameters. We suppose that parameter types are still Ontology *extended* classes. We consider at first the n-1 *Services*, then the general case of n-p Services.

Let's consider three classes "Patient", "MedicalFile" and "Institution", linked by relationships "hasFile" (one or several files for a given patient) and "managedBy" (only one institution for a given file), as indicated on the figure 6.

Let's consider a *Service* provided by an identity server delivering references to medical files for each ordered pairs (patient, institution) given as input, i.e.: S: (Patient, Institution) → $2^{MedicalFile}$. This is a 2-1 Service, with $In_1(S)$ = Patient, $In_2(S)$ = Institution, and Out(S) = MedicalFile. The relationship that links the output set {Files D} to inputs Patient P and Institution I is simply:

$R_S = hasFile^{True,\ PR\ (I)}$, where PR is the predicate defined by PR(I) = P.Institution = E

This is a *restriction* of the Ontology relationship "hasFile" by the predicate POST, parameterized by the other input parameter I. This may be written:

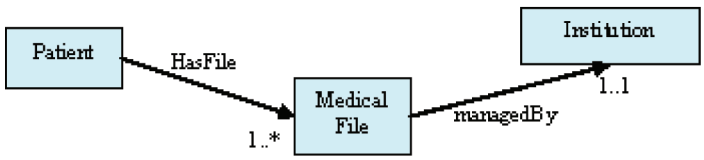
<MAP, $S_{in2\ =\ I,}$ hasFile, True, P.Institution= $In_2(S)$>.

where $S_{in2\ =\ I}$ is the 1-1 *Service* that associates to a patient P the output S(P, I), and where $In_2(S)$ denotes the value of the second parameter of S, i.e. the current value of input I.

In order to express the semantic of such a n-1 Service S: ($C_{in\ 1}$, …., $C_{in\ n}$) → 2^{Cout}, one has to define an algebraic formula of the form: R_S = EXPR ({R_k}, {PR_l}) that gives the *extended* relationship R_S linking the output with an i_0^{th} input parameter. This relationship is expressed by the means of Ontology relationships, and predicates {PR_l} involving the values of the other input parameters from $In_i(S)$, $i \neq i_0$, inside restriction predicates

The relationship R_S is the relationship *realized* by the Service S, i.e. such as <MAP, S, R_S>. It expresses the semantic of S on the basis of properties and predicates. This expression is not necessarily unique, in particular in the case where there exist in the Ontology *inverse* relationships of those used in the considered expression. The general method to define such an expression consists in: 1) determining a relationship path in the Ontology O linking one of the input parameter class to the output parameter class, and 2) adding predicates corresponding to the constraints involved by the data of other input parameters. The conditions of existence of such a mapping should be studied in detail.

Figure 6. Case of a mapping with (2, 1) service



Now, let's consider the general case of a n-p Service, with n input and p output parameters, i.e. S: ($C_{in\ 1}$, …, $C_{in\ n}$) → ($2^{Cout\ 1}$, …, $2^{Cout\ p}$). The semantic of the *Service* S is perfectly defined by the semantic of the p partial *Services* S_j: ($C_{in\ 1}$, …., $C_{in\ n}$) → $2^{Cout\ j}$ (p projections of S on the p output sets $C_{out\ 1}$, …, $C_{out\ p}$), which are n-1 Services, and so, the previous results apply.

Quality of Service

It may exist several ways to *realize* a relationship, i.e. to evaluate a property by the means of *Services* invocations in response to a user query. E.g., in order to access to the complete medical file of a given patient, we may decide to address in parallel direct queries to each institution, via ad hoc *Services* provided by each of them. We may also decide to query first a relevant *Service* provided by the regional health server, that will deliver the set of institutions in which the given patient has a medical file, then to request only the relevant ones. This choice is influenced by many factors such as the number of institutions, the expected delay of execution of each individual *Service*, their average availability, and, may be, some additional factors such as the expected quality of data, factor that gives higher quality to fresh data against data with possible lack of recent pieces of information (e.g. data from mirror or cache sources). Each factor may be quantified with a magnitude relevant with its meaning (e.g. a time, a probability, etc.). We consider here a set of quality factors F_i, i=1, …, p, with their associated metrics q_i. We consider the *quality function*:

$$q = [q_1, .., q_p] = Q(S_i),$$

that associates to each *Service* S_i, a p-dimension vector where the i^{th} dimension is the quantification of the F_i factor. Let be a *function of preference*:

$$Pref = \Lambda(q) = \Lambda([q_1, .., q_p]) = \sum_{i=1, \ldots, p} \alpha_i . q_i$$

provided by a calling entity, and that aggregates the various quality dimensions is a single relevant value and enables to compare various *realisations* of a given evaluation. For each factor q_i, we should express rules that define how to aggregate values of q_i factors when *services* are combined by composition or by union:

$$q(S_1 *\ S_2) = F(q(S_1), q(S_2)) = [F_i\ (q_i(S_1), q_i(S_2))]$$

$$q(S_1 + S_2) = G(q(S_1), q(S_2)) = [G_i(q_i\ (S_1), q_i(S_2))]$$

where F_i and G_i functions depend on the semantic of the considered q_i factor. Depending of this semantic, each F_i or G_i function may be a sum, a maximum, a minimum, etc. In the case of composition, where the number of S_2 invocations depends on the cardinality of S_1 results, the definition of F_i should reflect the chosen strategy of optimization (e.g. minimax). So, the *Services* infrastructure should provide relevant meta information such as the maximum or the mean cardinal of results for each *Service*. As a simplification, we may write:

$$q(S_1 *\ S_2) = q(S_1) *\ q(S_2)$$

$$q(S_1 + S_2) = q(S_1) + q(S_2)$$

to denote the combination of QoS vectors by the $*$ and + operators.

Automated Execution Plans Generation

We study here the general issue of determining the set of *Services* that should be invoked in response to a request, as well as the way in which they have to be orchestrated to meet their objective. Firstly, we define in details the various elementary issues. Then we present the principles of new *mapping* assertion generation that may be followed to solve our problem. This enables

to present at last an original algorithm providing solutions to our problem.

The Execution Plan Problem

So far, we have defined: 1) a way to express the semantic of the various *Services* provided by a given configuration, and 2) a way to express user queries under the form of *derived* properties evaluations. Such properties are expressed with the use of a combination of the various *native* elementary properties of the ontology.

Now, the main question is to determine the relevant execution plan of *Services* invocations that will deliver the expected result, i.e. the transformation of a given algebraic expression of properties into a plan of *services* invocations. Such a plan will be called here an *orchestration* plan. As the order of invocations is significant, and as some *Services* may be invoked in parallel, such an *orchestration* plan may be represented by an *execution graph*, where nodes represent the intermediate results, and where branches represent (sequential and/or parallel) tasks to execute.

In order to simplify the presentation, we focus here the presentation on the evaluation of *derived* object properties (relationships), the whole approach described in this section remaining valid for the evaluation of datatype properties (attributes), with a simple extension.

Having a configuration defined by an ontology, a set of defined *derived properties*, and a set of *Services*, with definitions of *mappings* relating *Services* to ontology properties, we consider a property R to evaluate, with a possible given *utility function*, and an individual x given as input. There are in fact three problems:

- A first issue is this of determining whether, in the given configuration, there exists or not a plan of *Service* invocations that may deliver the expected values.
- If we can be sure there exists a solution, a second issue concerns the construction of the solution graphs, and, if there are several solutions, the determination of the optimal plan regarding the criteria defined by the *function of preference*.
- If there is no solution, a third issue is this of determining possible restricting conditions regarding the input x, in case of which it would exist partial solutions

The second problem is this of the search for a *uniform optimal solution*, i.e. a unique solution which is optimal for any input x. If this solution exists, this will define an optimal *orchestration* plan that may be kept in memory for any further evaluation <R, x, ?y> to perform on this property R (static optimal execution plan). If it does not exist, there may exist solutions which are optimal for some $(dom(R))^p$ subclasses of dom(R).In this case, at runtime, the plan to execute should be the relevant one regarding the value of the input (dynamic execution plan).

In this the following, we present an approach that enables to deal with the three issues, thanks to one single algorithm. Such an algorithm generates a set of possible execution graphs. An execution graph determines a plan for *Service* invocations, with some *Service* composition (execution of two *Services* as a sequence), union (concurrent execution with join and result fusion), and restrictions (invocation with test condition). An execution graph G_i defines an algebraic combination of *Services*. During the execution of the algorithm, we shall consider plans that *realize* the property to evaluate, but also plans that *realize* only a subpart of the expression.

An execution graph G_i has an origin denoted $Orig(G_i)$, which stands for an input set of individuals, labeled by a $PRE(G_i)$ predicate, expressing the conditions to be satisfied by the input for the plan to be valid. The execution graph has also a end denoted $End(G_i)$, that stands for the output set of individuals, labeled with a $POST(G_i)$ predicate, that can express limitations in the delivered results. An execution graph is also labeled with a QoS value $Q(G_i)$.

An execution graph G_i which have the dom (R) class as origin, the range(R) class as end, and verifying POST(G_i) = True, will be called a *candidate partial solution*. If, in addition, PRE = True, then G_i will be a *candidate solution*. If we want to evaluate in advance an *orchestration* plan associated to a property, we shall apply at first the algorithm. Then, if there are candidate solutions, then the delivered solution (uniform optimal solution) will be the candidate solution G_0 maximizing QoS(G). This plan will enable to evaluate the property for any value of the input individual x. If there is no *candidate solution*, but if there exists some *candidate partial solutions* $\{G_j\}$, then it will be possible to evaluate the property iff the input given individual x satisfies the PRE(G_j) predicate. The delivered solutions will be the *partial candidate solution* G_1 maximizing QoS(G_j) among all partial candidate solutions such as PRE(G_j) (x) is satisfied. At the contrary of the previous case, there is here only a pseudo order on solution, as their associated valid input domains are not the same. This pseudo order become a total order as soon as the input individual x is specified. In this case, for a given input x_0, either there will be no solution; or there will be a solution for which the optimality is ensured, only for x_0.

Orchestration of Services

The basic idea on which our algorithm is based will be this of an iterative generation of new *mapping* assertions, derived from already defined ones. The problem is this of identifying the states in which it is actually possible to generate such new *mappings assertions*. Let be two relationships R_1 and R_2, and two *services* S_1 and S_2, such as:

<MAP, S_1, R_1, PRE_1, $POST_1$> and <MAP, S_2, R_2, PRE_2, $POST_2$>

In order to characterize $S_1 *\ S_2$ and $S_1 + S_2$ in terms of *mappings*, we use the following properties:

Property 1: It is possible to define a *mapping* for $S_1 *\ S_2$ iff: $POST_2$ = True AND PRE_1 = True, and this *mapping* is:

<MAP, $S_1 *\ S_2$, R_1* R_2, PRE_1, $POST_2$>

As a particular case, we may notice that, if S_1*realizes* R_1 (PRE_1 = $POST_1$ = True) and if S_2*realizes* R_2 (PRE_2 = $POST_2$ = True), then $S_1 *\ S_2$*realizes* $R_1 *\ R_2$

Property 2: the best *mapping* that may be defined for $S_1 + S_2$ is:

<MAP, $(S_1 + S_2)$, $(R_1 + R_2)$, PRE_1 OR PRE_2, $POST_1$ AND $POST_2$>.

So, the *mapping* exists iff $POST_1$ AND $POST_2$ ≠ False, i.e. iff out(S_1) ∩ out(S_2) ≠ Ø.

Example: let be a class C_1 with a datatype property A, a class C_2 with a datatype property B, a relationship R_1 from C_1 to C_2, and a relationship R_2 from C_2 to C_3.If we have some *Services* $S_{1,1}$, $S_{1,2}$, $S_{2,1}$, $S_{2,2,}$ such as:

<MAP, $S_{1,1}$, R_1, $PRE_{1,1}$ = (A=a_0), True> ; <MAP, $S_{1,2}$, R_1, $PRE_{1,2}$ = (A≠a_0), True>

<MAP, $S_{2,1}$, R_2, $PRE_{2,1}$ = (B=b_0), True> ; <MAP, $S_{2,2}$, R_2, $PRE_{2,2}$ = (B=b_0), True>

At first, we may generate two *mappings*. The first one is: <MAP, $S_{1,1} + S_{1,2}$, R_1, True, True> and the second one is: <MAP, $S_{2,1} + S_{2,2}$, R_2, True, True>. Considering this new *abstract Services* S = $S_{1,1} + S_{1,2}$, and S' = $S_{2,1} + S_{2,2}$, it appears that we may generate a new *mapping* involving S, that is: <MAP, S, $*$ S', R_1, True, True>. At the contrary, in the case we would have at the beginning the following *mappings*:

<MAP, $S_{1,1}$, R_1, $PRE_{1,1}$ = (A=a_0), $POST_{1,1}$ = "B=b_0">

<MAP, $S_{1,2}$, R_1, $PRE_{1,2}$ = (A≠a_0), $POST_{1,2}$ = "B≠b_0">

the only *mapping* to be derived is: <MAP, $S_{2,1}$ + $S_{2,2}$, R_2 >, and no other new *mapping* may be generated after that. This is symbolized on the Figure 7.

Before presenting the algorithm, we state the following definitions:

Definition (service equivalence). Given two Services S_1 and S_2, whose semantics are defined by the two mapping assertions <MAP, S_i, R_i, PRE_i, $POST_i$>, i = 1, 2. S_1 and S_2 are said to be equivalent ($S_1 \approx S_2$) iff $R_1 = R_2$, $PRE_1 = PRE_2$, and $POST_1 = POST_2$.

We shall say that $S_1 \geq S_2$ iff $R_1 = R_2$ AND ($PRE_1 \rightarrow PRE_2$) AND ($POST_1 \rightarrow POST_2$) (i.e. S_1 is a better realization than S_2 of the same relationship $R = R_1 = R_2$)

It has to be noticed that "≥" is <u>not</u> a pseudo order, because $S_1 \geq S_2$ AND $S_2 \geq S_1$ implies $S_1 \approx S_2$, but not necessarily $S_1 = S_2$. In an algebraic expression of *Services*, it will be possible, at first, to replace an operand *Service* S_i by an equivalent *Service* S_j or by a *Service* S_j such as $S_i \geq S_j$ if $q(S_j) \geq q(S_i)$. In order to *realize* an algebraic expression of properties, it is then necessary to find in the repository of available *Services*, all the *Services* that *realize* (totally or partially) a part of the expression. The *Services* found in the repository, that will "match" a given relationship, will be the possible building blocks of a future execution plan for the evaluation of R.

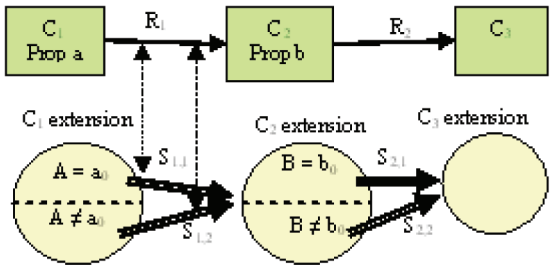
At last, we define the *matching* of *Service* with the following definition.

Definition (service matching). Given a relationship R defined as an algebraic expression, given a Service S, we shall say S matches R iff if exists a mapping assertion such as <MAP, S, R', PRE, POST>, where R' is a sub expression of R.

An Algorithm for Execution Plan Generation

We present here the algorithm enabling the construction of solution execution graphs in response to a given *derived* property evaluation. Depending on the case, the algorithm will provide: 1) A *uniform optimal plan* that will work for any individual x from the class dom (R), or 2) a set of plans with a associated constraints on input and QoS values for each plan.

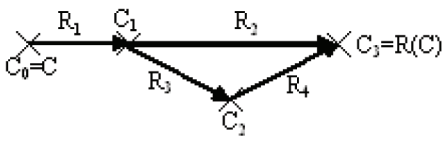
Figure 7. Example of mapping generation



The algorithm works in five main steps:

- **Step 1. Evaluate the input algebraic expression in terms of elementary *native* properties**. We replace, in a recursive way, each operand *derived* property by their definitions. So, for an expression such as: R = R' + R'', where R' and R'' are *derived* properties defined by: $R' = R_{1*}R_2$, $R'' = R_{1*}R'''$, and $R''' = R_{3*}R_4$, and where R_1, R_2, R_3 and R_4 are *native* properties. The expression of R will be transformed via the following iterations: $R = R_{1*}R_2 + R_{1*}R'''$, then $R = R_{1*}R_2 + R_{1*}R_{3*}R_4$
- **Step 2. Simplify the algebraic expression**, thank to algebraic properties of operators (factorization). So, the expression: $R = (R_{1*} R_2) + (R_{1*} R_{3*} R_4)$ becomes: $R = R_{1*} (R_2 + (R_{3*} R_4))$. This is done in order to minimize the number of *Service* invocations and the flow of intermediate results. The algebraic formula is stored under the form of an *execution tree*, where the leaves are operands R_i and the nodes are partial results. In the above example, we shall have the following nodes: (N_1):$R_{1*} (R_2 + R_{3*}R_4)$; (N_2): $R_2 + R_{3*}R_4$; (N_3): $R_{3*}R_4$

Figure 8. Example of flow – relationship graph



- **Step 3. Build the *flow - relationship graph* associated to the expression**. On the basis of the previous, maybe simplified, expression, one generates a directed acyclic graph, corresponding to the evaluation of the result property, where nodes $\{C_i\}$ stands for collections of values corresponding to the various intermediate levels of evaluation, and edges $\{R_j\}$ are instances of relationships, relating an input collection $C_{j,1}$ to an output one $C_{j,2}$, and labeled by the corresponding operand relationship.

The origin of the graph is the input x issued from the class $C_0 = dom(R)$, the end of the graph is the expected collection of result values (it has to be noticed that the same relationship may have several instances as several distinct edges in the graph). As an example, the expression $R = R_1 * (R_2 + (R_3 * R_4))$, above considered, generates the following *flow - relationship graph*, which has four nodes and four edges.

- **Step 4. Find Services that matches parts of the graph**. This step consists in extracting from the repository of *Services*, all the *Services* that match a part of the *flow – relationship graph*, as it has been defined in the previous section (*match* operator). We get a subset $\{S_i\}$ of *Services* which will be the input *service* set of the following step of the algorithm.
- **Step 5. Generate the candidate execution graphs**. This is done by combining the selected *Services* in various manners, in order to construct a combination of *Services realizing* the relationship R, if this realization exists. There are two possible approaches for developing such an algorithm: the first one is this of a *descendant* algorithm that start from the relationship to evaluate, and tries to express it in function of the given input *Services*. The second one, that we have adopted here, is this of an *ascendant* algorithm. The algorithm takes the input *Services*, and combines them iteratively, in order to derive at each iteration new *mappings* that give *better* or *more complete* realizations (in the meanings of the QoS and of the above defined comparator ≥) than those already exhibited. The algorithm stops when there is neither new possible *matching*, neither new (better, or more complete) *mapping*. This stop is guaranteed due to the strict increasing of a function on a discrete set. When there exists a *mapping* involving both the graph origin and end, then there exists an execution plan which is at least a partial candidate solution.

If the *uniform optimal solution* does not exist, this algorithm stops with, as present state, the best partial solutions, solving de facto the third issue presented at the beginning of this section. With no global solution, these partial solutions will nevertheless permit to have a possible available solution for a given input x. In this case, the best solution will be selected at runtime.

The principle of the present step of the algorithm is so the following: we consider now the *realization graph*, that is a directed graph based on the previous flow – relationship graph, where nodes are those of the *flow – relationship graph*, but with possible additional edges. So, there are two types of edges in this graph:

- The *relationship edges*, that are the edges of the *flow - relationship graph*, standing for operand relationships,

- New edges iteratively generated by the algorithm, and standing for partial *realizations* of the *flow - relationship graph*. Such a *Service edge* may represent a *real Service*, as well as an *abstract Service*, i.e. an algebraic expression of some real *Service*s. A *Service edge* (C_i, C_j) is a *realization* of the relationship relating C_i to C_j. It is labeled by a 3-uple (PRE, POST, q) where PRE and POST are *mapping* predicates, and q is the QoS vector associated to the realization.

The *relationship edges* are present at the beginning of the algorithm. The *Service edges* are incrementally added at each stage of the algorithm. The algorithm corresponding to this step may be expressed as a recursive procedure: at each stage, a new *Service* is considered. This *Service* may come from the input *Service* set or may have been generated at a previous stage. We integrate this *Service* as a new edge in the *realization graph*, labeled by the existing *mapping*. Among the already present *Services edges*, we consider those that may be combined with the new *Service* to generate at least a new *mapping*. In case it is possible, only one new *Service edge* is created, and a similar process is applies recursively. This recursive algorithm is so:

```
Algorithm
  for each S IN Input Service set
    Express mapping M_i = (R, PRE, POST)
    between S and a Relationship R
    Add a new Service edge in the graph
labelled
    with this mapping M_i and the QoS vec-
tor of S
    INTEGRATE (S)
  End for each
end Algorithm
Procedure INTEGRATE (in S: Service)
if There exists S' such as (S' ≥ S) OR
((S' Eq S) AND q (S')  ≥  q (S)
then delete S
else Associate S to possible Service
Edges E_i for each E_i
Determine the associated mappings (R_i,
PRE_i, POST_i) and the resulting quality of
service q_i
Add a Service edge labelled with R, PRE,
POST and q
INTEGRATE (E_i)
end for each
end if
end Procedure
```

Redundant *Service edges* (i.e. that correspond to *Services inferior* to an already present *Service*, or *equivalent* with lower QoS) are removed in order to avoid the explosion of non significant *mappings*. At the termination of the algorithm, the partial solutions are the *Service edges* (if they exist) linking the origin with the end of the graph, and labelled with a (POST = True) condition. If one of such *Service edges* has a (PRE = True) condition, then it is an optimal solution. If not, the result of the algorithm is the set of 3-uples (S_i, PRE_i, q_i), where S_i is a partial realization, PRE_i the corresponding PRE validity condition, and q_i the associated QoS. In any case, a solution is an algebraic expression of the input *Services* that defines an orchestration of *Services*, i.e. an execution plan defining the *Service* invocation to execute with sequences and possibly concurrent branches, as well as test conditions to perform. The solution to the problem of a datatype property evaluation is based on the evaluation of a relationship, as seen above, with some additional specific operations not detailed here.

A Software Framework

On the basis of our approach, a software framework has been prototyped. This framework supports the definition and management of *derived* properties, issued from various user communities, and defined on the basis of a provided ontology defined via

PROTEGE. It supports the generation and run time execution of the relevant *Service* orchestration in response to a request for information. The framework also enables the management of Web *Services*, with all their relevant associated meta information. The framework includes: 1) A derived properties repository, where the description of derived object and datatype properties are stored, and queried, 2) a Web Service platform, that enables to develop and execute Web Services, with interface types conformant with classes and datatypes of the ontology, 3) a Web Service repository that stores the semantic descriptions of Services, and other required meta information, as defined in the present paper, In addition, we have two software components: 4) the Generator that generates execution plans, with an implementation of the algorithm we have proposed, and the Orchestrator, that executes such generated plans.

The framework has been tested in the context of the federation of several Health Information Servers, described in the Case Study section, and more specially in the context of a new application in oncology: a support to multidisciplinary meetings in which therapeutic decisions are taken. The results are very encouraging because the framework clearly adds important factors of openness and flexibility to the context. The experiment shows the approach constitutes an efficient way to easily integrate new information sources in an Information Server federation, and take into account new user needs, while avoiding huge amount of specific software coding.

FUTURE RESEARCH DIRECTIONS

To deal with the problem of Web Services automatic discovery and composition, we have presented in this chapter an Algebra allowing rigorous combinations of Ontology properties. This algebra enables to attach a precise meaning to any expected piece of information, as well as to confer to an existing Web Service a well defined semantic based on the Ontology concepts. On this basis, we show it is possible to develop an efficient algorithm generating optimal execution plans of Web Services.

The main hypothesis on which relies the applicability of the approach is this of a common agreement of user communities on an exhaustive and fine-grained ontology of their domain. Of course, this is at present a major limitation in the adoption of such an approach, but we do think that, on one hand, a capture of the application domain via an Ontology, and, on the other hand, a rigorous model, able to confer a well defined formal semantic to a Service, are the absolute requisite to achieve the expected objective. There are still many difficult issues to solve in the future in order to meet the complete objective of automated discovery and composition. To continue in the direction presented in this chapter, three main axes may be defined now:

A first axis consists in extending our approach to more general form of ontology properties. In particular, we may consider those that would be *deducted* by the means of inference rules. For example, such rules would be defined by permitting cyclic definitions of derived properties. This would introduce a reasoning aspect in the approach, and would lead to logical approaches for orchestration plan generation;

A second axis concerns the extension of the approach by considering Services having an *effect* on internal data (creation, update) and/or on external world. This would permit to address topics related to popular applications, such as these of electronic commerce, or construction of ad hoc processes in Information Systems of companies. Substantial works have already be done on these issues, and we think an Ontology-based algebraic approach would bring new developments;

The third axis is this of elaborating on the results in the framework of present standards and languages (OWL, OWL-S, …). In particular, declarative languages and user-friendly tools adapted to the problematic would be highly required. In

addition, approach for the reuse and integration of existing Web Services in an Ontology-based approach is also a challenging issue.

CONCLUSION

The flexible integration of heterogeneous information sources, as well as the ways to query them by the mean of Web Services orchestrations, are not recent issues. But with the increasing importance of Ontologies, new approaches have to be developed. In the context of Semantic Web and widely spread decentralised architectures, a new challenge relative to the taking into account of the semantic dimension has now appeared with a very strong importance. This dimension concerns in particular the definition of semantic correspondences between on one hand Web Services, and, on the other hand, knowledge about domain, expressed via Ontologies.

In this paper, arguing that an important part of Services semantic may be captured by the means of Ontology relationships, we show that, on the basis of this hypothesis, it is possible to build a consistent and well formalized Algebra that enables to perceive any property definition, combination and evaluation of them as algebraic operations.

In this context, we propose (1) an Algebra of Ontology properties, that enables definitions of new properties on the basis of native ontological ones, (2) a model that enables to associate a formal semantic to a Service using mapping assertions using Ontology properties, and (3) a general algorithm that performs an automated generation of execution plans, and translates a property evaluation into a optimal orchestration of Services.

On the basis of our proposed approach, a software prototype has been developed and deployed in the context of an Health Information Systems, in order to provide new facilities. The evaluation shows that our approach provides to the application the properties of openness and flexibility, saving huge efforts that would have been spent in specific code development in the case of a classical software development approach.

REFERENCES

Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schrnidt, M.-T., Sheth, A. & Verma K. (2005). *Web service semantics – wsdl-s*. W3C Member Submission, November 2005

Ayadi, N., & Lacroix, Z. (2008, November). Biomap: a deductive approach for resource discovery. In *UWAS'2008 - The Tenth International Conference on Information Integration and Web-based Applications Services*, 24-26 November 2008, Linz, Austria, pages 477-482

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific American*, (5): 35–43.

Gao, H. T., Hayes, J. H., & Cai, H. (2005). Integrating biological research through web services. *Computer*, *38*(3), 26–31. doi:10.1109/MC.2005.97

Goble, C., Kesselman, C., & Sure, Y. (Eds.). (2005). In Semantic Grid - Convergence of Technologies, number 05271 in Dagstuhl Seminar Proceedings.

Klusch, M., Pries, B., & Sycara, K. (2006). Automated semantic web service discovery with OWLS-MX. In *Proc. 5th International Joint Conference on Autonomous Agents and Multiagent Systems,* (pp. 915-922), Hakodate, Japan. New York: ACM.

Kopecky, J., Vitvar, T., Bournez, C., & Farrell, J. (2007). SAWSDL: Semantic Annotations for WSDL and XML Schema. *IEEE Internet Computing*, *11*(6), 60–67. doi:10.1109/MIC.2007.134

Kourtesis, D., & Paraskakis, I. (2008). Combining SAWSDL, OWL-DL and UDDI for Semantically Enhanced Web Service Discovery. In M. Hauswirth, M. Koubarakis, and S. Bechhofer, (Eds.), *Proc. 5th European Semantic Web Conference,* (LNCS 5021, pp. 614-628). Berlin: Springer.

Martin, D., Burstein, M., Mcdermott, D., Mcilraith, S., Paolucci, M., & Sycara, K. (2007). Semantics to Web Services with OWL-S. *World Wide Web (Bussum)*, *10*(3), 243–277. doi:10.1007/s11280-007-0033-x

Maximilien, E. M., & Singh, M. P. (2004). A framework and Ontology for Dynamic Web Services Selection. *IEEE Internet Computing*, 5.

Paolucci, M., Kawamura, T., Payne, T. R., & Sycara, K. P. (2002). Semantic Matching of Web Services Capabilities. In *Proc. 1st International Semantic Web Conference on The Semantic Web,* (LNCS 2342 pp. 333-347). London: Springer.

Papazoglou, M. P., Traverse, P., Dustdar, S., & Leymann, F. (2007). Service-oriented computing: State of the art and research challenges. *Computer*, *40*(2), 38–45. doi:10.1109/MC.2007.400

Roman, D., & Keller, U., H. Lausen H., De Bruijn, J., Lara R., Stollberg,M., Polleres, A., Feier, C., Bussler, C. & Fensel, D. (2005). Web Service Modeling Ontology. *Applied Ontology*, *1*(1), 77–106.

Sirin, E. (2003). Semi-automatic composition of Web services using semantic description . In *Proceedings of Web Services Modelling, Architectures and Infrastructures workshop in conjunction with ICEIS'2003, 12(8), 72-7*. Hendler J, Parsia B.

Sycara, K., Paolucci, M., Ankolekar, A., & Srinivasan, N. (2003, December). Automated discovery, interaction and composition of Semantic Web services. *Web Semantics: Science . Services and Agents on the World Wide Web*, *1*(1), 27–46. doi:10.1016/j.websem.2003.07.002