Softw*I*re Integration –

an approach to service composition through distributed representations of arbitrary-length business ontologies

Hendrik Ludolph

Faculty of Science
Computer science department (IIUN)

University of Neuchatel

A thesis submitted for the degree of

PhilosophiæDoctor (PhD)

2015 December

Abstract

The demand for Software as a Service is ever increasing. With this demand comes a proliferation of third-party service offerings to fulfill it. It thus becomes crucial for organizations to find and select the right service to be integrated into their existing tool landscapes. Ideally, this is done automatically and flexibly. The objective is to always provide the best possible support to changing business needs.

Modern integration standards, such as ESB, SOA, or BPI evolved to face this challenge, e.g., through loose coupling. However, they lack description capabilities to adequately capture service semantics, needed for adequate automated service selection and integration. They also heavily depend on human expert intervention. Next generation semantic techniques, such as SAWSDL, OWL-S, or WSMO aim at correcting this situation. They use ontologies, which are based on formal rules of inference. These techniques claim to extract and interpret the meaning of data, thereby leading to more reliable automated service selection and integration.

To us however, knowing and interpreting is more than imperative, rule-based manipulation of arbitrary symbolic expressions. Pure symbolic, either syntactic or formal-semantic, approaches will thus not provide the bridge between automation and flexibility. This is even more so in competitive environments. Here, changes in the input data of third-party services offerings can neither be anticipated, nor can technical control be exercised. Unknown conditions can thus only be processed to the extent they are predefined.

To overcome this contradiction, we investigate a hybrid symbolic/connectionist approach. To this end, we provide a framework for automated service selection based on high-level business requirements, described by ontologies. Thereafter, we explore a specific supervised artificial neural network topology called LRAAM. With it, the compositional structure of the ontologies is transformed into a distributed (i.e., reduced, micro-semantic) representation. It shall enable

the system to develop an “own” or “inner” representation of service descriptions, which are then used for similarity analysis. A tool (*OntoProc*) was developed as a proof of concept. With it, we conducted different experiments to explore the validity of the approach. The tool generated output data, which we submitted to significance tests.

Based on the experiments, the current LRAAM implementation is not a reliable alternative for service selection and integration. We showed that the LRAAM performs correct selection of services under specific parameters. However, for more complex input data, it does not yet yield the expected results. We discussed adjustments to the implementation to improve *OntoProc*’s performance.

Despite the inconclusive results, we are, nevertheless, convinced that purely symbolic approaches to automatic integration are too restrictive when independent third-party services are concerned. Flexibility without human intervention is beyond their capabilities. Encouraged by recent developments in the field of connectionism (e.g., Deep Learning), we adhere to the chosen research venue. We see it as part of a new paradigm of operating on large vectors (i.e., connectionism) to replace rule-based manipulation of symbols.

Résumé

La demande de logiciel à la demande (*Software as a Service* ou *SaaS*) est en constante augmentation. Afin de remplir cette demande, une prolifération d'offres de services de type *third-party* s'en suit. Pour les organisations il est crucial de trouver et sélectionner le service approprié pour l'incorporer dans le parc TI existant. Idéalement, cela est effectué de façon continue et flexible, l'objectif étant de toujours fournir le meilleur support possible.

Pour faire face à cet enjeu, des standards modernes d'intégrations tels que ESB, SOA ou BPI ont été conçus pour permettre un couplage faible (*loose coupling*) entre les services. Par contre, il manque à ces standards la capacité de bien décrire et capter la sémantique des services. Cette sémantique serait nécessaire pour une intégration automatique adéquate. L'efficacité de ces standards dépend aussi considérablement de l'intervention humaine. Une nouvelle génération de techniques sémantiques telles que SAWSDL, OWL-S ou WSMO vise à corriger cette situation en utilisant des ontologies. Ces dernières sont basées sur des règles formelles d'inférence, ayant l'ambition de représenter et d'interpréter le sens des données. Cela devrait mener à une sélection et intégration automatique supérieure de services.

Nous croyons cependant que connaître et interpreter requiert plus que la manipulation réglémentée et impérative d'expressions symbolique arbitraires. Selon nous, les approches purement symboliques, autant syntaxiques que sémantiques, ne permettent pas de combiner automatisation et flexibilité. Ceci est d'autant plus vrai dans des environnements compétitifs. Dans ces environnements, les changements d'offres de services ne peuvent être ni anticipés, ni gérés de façon contrôlée. Des conditions inconnues ne peuvent être traitées que dans la mesure où elles ont été prédéfinies.

Afin de surmonter cette contradiction, nous proposons une approche hybride symbolique/connexionniste. À cette fin, nous présentons un cadre conceptuel de sélection automatique de services, basée sur des descriptions de haut niveau des besoins d'affaires. Ces descriptions sont décrites par des ontologies. Par la suite, nous explorons une topologie spécifique de réseau neuronal artificiel, nommé LRAAM. Avec ce dernier, la structure compositionnelle des ontologies est transformée dans une représentation distribuée (i.e., réduite, micro-sémantique). Le LRAAM amène le système à générer une représentation "propre" ou "interne" des descriptions ontologiques de services. Celles-ci peuvent être utilisées pour des analyses de similarité. Un outil (*OntoProc*) a été conçu comme preuve de concept. Avec celui-ci, nous conduisons différentes expériences afin d'étudier la validité de l'approche. Les données produites par l'outil ont fait l'objet d'analyse de signification statistique.

Se basant sur les expériences conduites, nous ne pouvons pas conclure que l'implémentation de LRAAM utilisée est une alternative fiable pour la sélection et l'intégration de services. Nous démontrons que sous certaines conditions, le LRAAM produit une sélection correcte de services. Cependant, il n'est pas possible d'arriver à la même conclusion pour des données d'entrée plus complexes. Nous discutons des ajustements à faire à l'implémentation afin d'augmenter la performance de l'outil.

Malgré des résultats non-concluants, nous sommes toutefois convaincu que des approches purement symboliques d'intégration automatique sont trop contraignantes quant il s'agit des services de type *third-party*. La flexibilité sans intervention humaine se situe au-delà de leurs capacités. Encouragé par des développements récents dans le domaine du connexionnisme (e.g., Deep Learning), nous adhérons tout de même à la piste de recherche choisie. Nous le voyons comme faisant partie d'un nouveau paradigme, qui est de traiter des larges vecteurs pour remplacer la manipulation réglementée de symboles.

To ...

Acknowledgements

Contents

List of Figures

List of Tables

LIST OF TABLES

Glossary

ANN	Artificial Neural Network, page 6
API	Application Programming Interface, page 20
BPEL	Business Process Execution Language, page 40
BPI	Business Process Integration, page 23
CI	Configuration Item – an asset, service component or other item controlled by the configuration management process, stored within the configuration management database., page 1
COBIT	Control Objectives for Information and related Technology, page 2
COM+	Component Object Model Plus, page 23
CORBA	Common Object Request Broker Architecture, page 23
COTS	Commercial Off-The-Shelf, page 2
DCOM	Distributed Component Object Model, page 23
DOLCE	Descriptive Ontology for Linguistic and Cognitive Engineering, page 65
EAI	Enterprise Application Integration, page 19
EI	Enterprise Integration, page 19
ESB	Enterprise Service Bus, page 38
IaaS	Infrastructure as a Service, page 17
IDL	Interface Definition Language, page 30
IoT	Internet of Things, page 67
ITAM	IT Asset Management, page 1
ITIL	Information Technology Infrastructure Library, page 1
JMS	Java Messaging Service, page 23
JSON	Javascript Object Notation, page 23
LRAAM	Labelled Recursive Auto-Associative Memory, page 8
MOM	Message Oriented Middleware, page 31

LIST OF TABLES

ODBA	Ontology based data access, page 9
OOTB	Out Of the Box, page 2
ORB	Object Request Broker, page 29
OWL	Ontology Web Language, page 9
OWL-S	Web Ontology Language for Web Services, page 55
RMI	Remote Messaging Invocation, page 23
RPC	Remote Procedure Call, page 28
SaaS	Software as a Service, page 2
SAWSDL	Semantic Annotations for WSDL and XML Schema, page 55
SOA	Service Oriented Architecture, page 20
SOAP	Simple Object Access Protocol, page 32
UDDI	Universal Description, Discovery, and Integration, page 33
WSDL	Web Service Description Language, page 32
WSMO	Web Service Modelling Ontology, page 55
XML	Extensible Markup Language, page 23
XSLT	Extensible Stylesheet Language Transformation, page 33

1

Introduction

An organization undergoes constant change. The pressure to do so is brought about by competition, internal politics, organizational evolution, technical progress, and cost containment. An IT organization must support this change. Information Systems (IS), i.e., a specific assembly of applications or services[1] to support distinctive organizational needs (Izza 09), play a major role in this endeavour.

1.1 Research context

Generally, corporate IT organizations use a wide range of IS. They are composed of many applications and services to maintain IT operations. For example, such applications are deployed for:

- managing (e.g., discovering, inventorying) network attached devices, such as laptops, servers, or workstations; or
- managing (e.g., storing, tracking) *Configuration Items* (CI) (ITS 07) and relationships among them, or
- managing (e.g., logging, dispatching) IT-related incidents and problems.

Often, the activities are part of good-practice standards for IT operations, e.g., *Information Technology Infrastructure Library* (ITIL) (ITS 07), *IT Asset Management* (ITAM) (Int 08), or

[1]The relationship between application and service is explained below.

Control Objectives for Information and related Technology (COBIT) (IT 07). Software vendors align their product portfolios to meet these standards, seeking economies of scale. Consequently, applications often follow a “Commercial Off-The-Shelf” approach (COTS). By the same token, organizations adopt IT standards and expect lower costs through “out-of-the-box” implementations (OOTB). For example, ITIL defines a process “Service Catalog Management,” with associated tasks, such as providing, updating, deleting requestable catalog items. A variety of service catalog management applications are offered within the market. They are tailored to support these standardized processes and tasks. However, they may vary as to how functionality is implemented.

Furthermore, the service catalog management process is linked to other management processes, e.g., change, configuration, or IT financial management[1]. Again, an organization may choose from a considerable number of applications tailored to support these processes, or activities thereof.

In the era of *Cloud*, this number increases even further. Organizations, such as SalesForce, ServiceNow, Akamai, etc. emerged to extend on-premise application offerings to off-premise Cloud-based service offerings, i.e. Software as a *Service* (SaaS). Services can represent anything from a simple request to complicated business processes (Lehaney *et al.* 11). They can participate in many different IS (Kale 14). The objective is to increase reusability of applications and lower integration hurdles through standardization.

The above described organizations thus commoditize and commercialize services, such as Customer Relationship Management, IT Service Management, Perfomance & Availability. Other organizations request, contract, and integrate these services instead of setting up the functionality by themselves. No expensive know-how for the service is needed. If later the service is no longer useful, the contract is cancelled. The service, technically and commercially, disappears. Clearly, this flexibility appeals to more and more organizations these days to improve their IS (Cisco 14). Some authors even claim that it becomes mandatory for keeping a competitive advantage (Fensel and Bussler 02, Hussain and Mastan 14).

The resulting commercial success of organizations like SalesForce with so called super-normal profits, however, acts as an incentive for new firms to enter the market (cf., (Frank and Cartwright 13, Chap. 11) – *long-term zero-profit equilibrium*) and copy the commercially successful offerings. This necessarily leads to a proliferating number of *similar* Cloud-based services to choose from as part of an adequate IS (Bughin and Chui 10).

[1]For more details, the interested reader is referred to respective sources on ITIL.

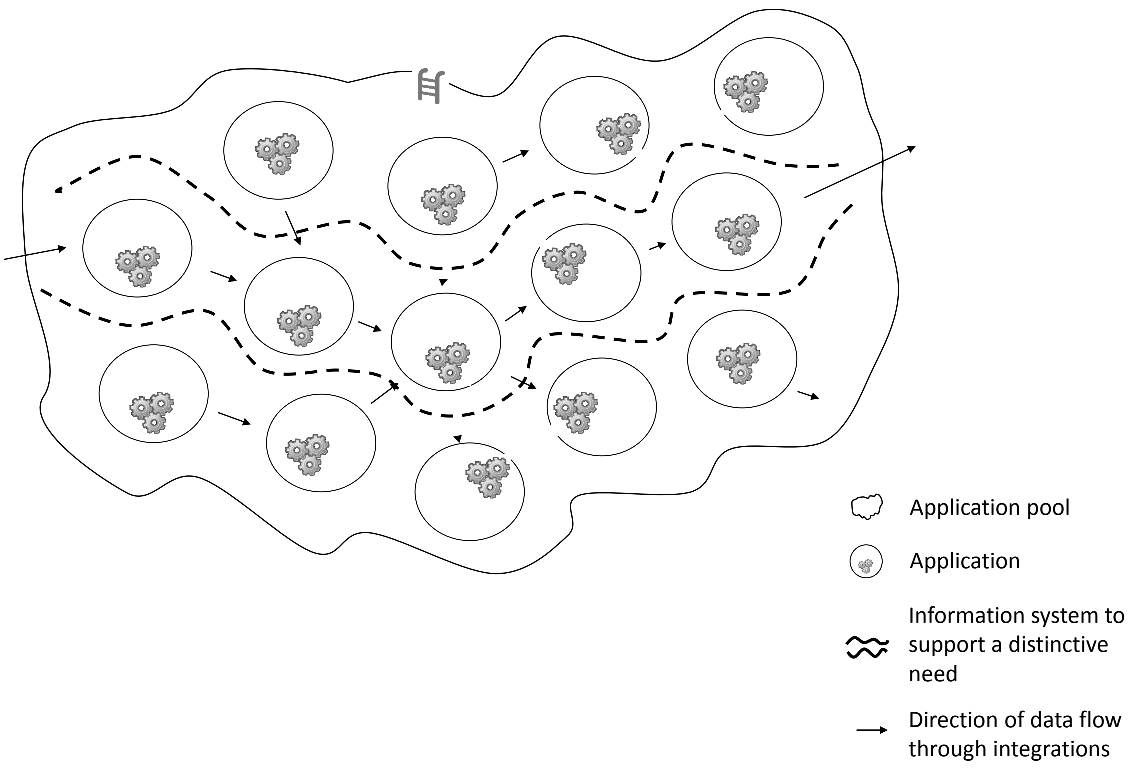

Application pool
Application
Information system to support a distinctive need
Direction of data flow through integrations

Figure 1.1: Application, application pool, and information system. The arrows indicate a directed stream of data from one application to the next.

Eventually, the adoption of IT standards and resulting commoditization of services leads to pools of on-premise applications and Cloud-based services, working in concert within corporate IT landscapes. Within such pools, a given application or service is integrated with one or more other applications or services with respective data input and output interfaces (cf., Fig. 1.1).

1.2 Problem statement and implication

Despite the above technical and process standardization efforts, such as ITIL, challenging issues can be identified:

- IT architectures evolve due to changing business needs and the need for IT-business alignment (Singh and Huhns 05, Izza 09, Panetto and Cecil 13).

- Stable corporate organizations are political in nature with special-interest groups forming within. Indeed, following (Olson 82), stable organizations with unchanged boundaries tend to accumulate more special-interest groups for collective actions[1] over time. Competing software providers use this organizational behavior to attach to such groups to increase their chances of selling software products (applications or services). They seek replacement of other groups' favorite, mostly competitors' products. This may or may not lead to efficiency gains. It certainly leads to changes within an organization's application and service pool.

- Data quality (e.g., global naming conventions) in general and because of the former point is a difficult task for organizations (Hüner *et al.* 09). Decentralized setup often entails increasing syntactic and semantic divergence not accounted for within existing application or service interfaces.

- Following a vendor–driven innovation and selling strategy, applications or services are upgraded to more reliable versions, or because they fall out of support.

These points summarize the problem domain on which we base our entire discussion. Namely, organizations need to constantly assess on-premise applications' or Cloud-based services' adequacy to support the business. Based on expensive analyses, applications or services (henceforward subsumed as services) are evaluated, and thereafter either dismissed, reconfirmed, or newly selected. If a new service is selected, continued human supervision, manual design, and costly adjustments of service interfaces are needed.

1.3 Research motivation

As depicted in Sections 1.1 and 1.2, technological outfit and corporate social behavior are intrinsically connected. The latter implies conflicting interests within and outside corporate organizations. Those interests are reflected in organizational structures, processes, and ideologies. Organizations have to cultivate a plurality of ideas, management approaches, and hence a selection of methods, processes, and technologies (Brunsson 02). Indeed, these days, corporate IT organizations constantly need to navigate among the IT delivery and operation poles

[1]Any action taken together by a group of people whose goal is to enhance their status and achieve a common objective. Cf., http://www.britannica.com/EBchecked/topic/1917157/collective-action-problem.

On-Premise, *Public Cloud*, *Private Cloud*, *Hybrid Cloud*, *Managed Service*, *Outsourcing*, or *Outtasking*[1].

Organizations' and their IT departments' service pools are thus constantly changing and a significant amount of money is spent to assess and disconnect "undeserving" services and select and connect "the latest and greatest" new ones. An arsenal of highly trained, highly paid, internal or external consultants are hired to do this never-ending job, namely *selection* and *integration* of COTS services.

Indeed, the selection process involves a lot of challenges. It is not at last the high complexity of the process itself (Ruhe 03), especially due to organizational agendas and politics mentioned above. Following (Mohamed *et al.* 07), therefore no "silver bullet" selection method exists. To the authors, different approaches have different effectiveness depending on the context. As for the integration part, Gartner (Lheureux 12) predicts that by 2018, more than 50% of the cost of implementing 90% of new large information systems will be spent on integration. Specifically, it consists of bringing data from one service together with that of another service (Stimmel 01). Integrators have thus to be concerned with the purposeful sharing of data among those services and data sources in the organization (Linthicum 99). The purpose is then to support the flow of data as part of corporate business processes. These in turn represent the very lifelines of an organization.

In this work, we pursue an approach towards automation of at least parts of the mentioned selection and integration efforts. We want to devise and analyze a method for a more intelligent Service Selection and Composition (SSC) under changing conditions.

Such an SSC should (1) understand what "good" means; (2) automatically select the best[2] service; (3) automatically integrate the selected service; and (4) do this continuously. Note that the notion "select" goes beyond simple "discovery." It represents a degree of intelligence, namely identification and analysis towards synthesis of possible actions (Zdravković *et al.* 14). We seek to substitute or at least support the usual time consuming manual, complex selection process, the translation of business requirements into technical requirements, and the subsequent implementation of interfaces among the selected services through labour-intensive integration projects. **Eventually, our goal is to save organizations money**.

Current industrial integration techniques, such as traditional middleware (e.g., remote procedure call mechanisms, data oriented, component oriented, message oriented, application

[1] A description of these concepts is outside the scope of this document.
[2] The notions of "good" and "best" are defined in Chapter 2.

servers), EAI tools (e.g., MS Biztalk, Tibco), BPM (e.g., BPEL, BPMN), or SOA (Oracle Service Bus), are not suited for such a method. For many authors (Hoang and Le 09, Izza 09, Fensel *et al.* 11, Hoang *et al.* 14), these integration techniques are blind to the most crucial aspect of a more intelligent SSC: understanding the semantics of services. They merely regulate information and focus on meta-data exchange. They are syntactic in nature. The same authors therefore propose the *Semantic Web* approach, which enables machines to understand the meaning, or else, semantics, of services.

To realize such an SSC, we analyse the combination of two techniques. These techniques represent and manipulate data, information, and knowledge at the semantic level. Specifically, they are *Ontologies* and *Artificial Neural Networks* (ANN).

1.3.1 Ontologies

Following (Born *et al.* 07) and (Fensel *et al.* 11), semantic descriptions of services are necessary in order to establish and warrant interoperability that does not require a human to manually effect certain integrations that will be rapidly obsolete, or non-reusable in an environment with the dynamic contexts presented above (Fig. 1.2).

Therefore, in the present case, semantic descriptions are produced of business activities and how they are related, and of independent – competing – services (data model and functionality). Note, that a distinction between *Service* and *Web Service* is made in (Fensel *et al.* 11). For (Fensel *et al.* 11), a Service provides value, a Web Service represents a technical means to access it. Our focus is on *Service*, i.e., on what it does, as opposed to on how to access it technically.

Our assumption is, first, that service and business activity descriptions can be more or less *similar* and second, that similarity is a selection criterium. For our context, it means there is a *distance* that is minimized to obtain maximal similarity. In general, such a distance can be defined and measured in different ways. An example is the *Hamming distance*. To obtain it, one counts the minimum number of letter substitutions required to transform one string into another string – the fewer the substitutions, the more similar the strings. For example, for two strings a = 'ibm' and b = 'hal', $d_{Ham}^{a,b} = 3$. For our context, the concepts of "distance" and "similarity" are introduced in greater detail in Section 6.3.

The most similar service (out of many) for a certain business activity is thus selected. Eventually, the selected services' integration is accomplished by adequate low-level attribute

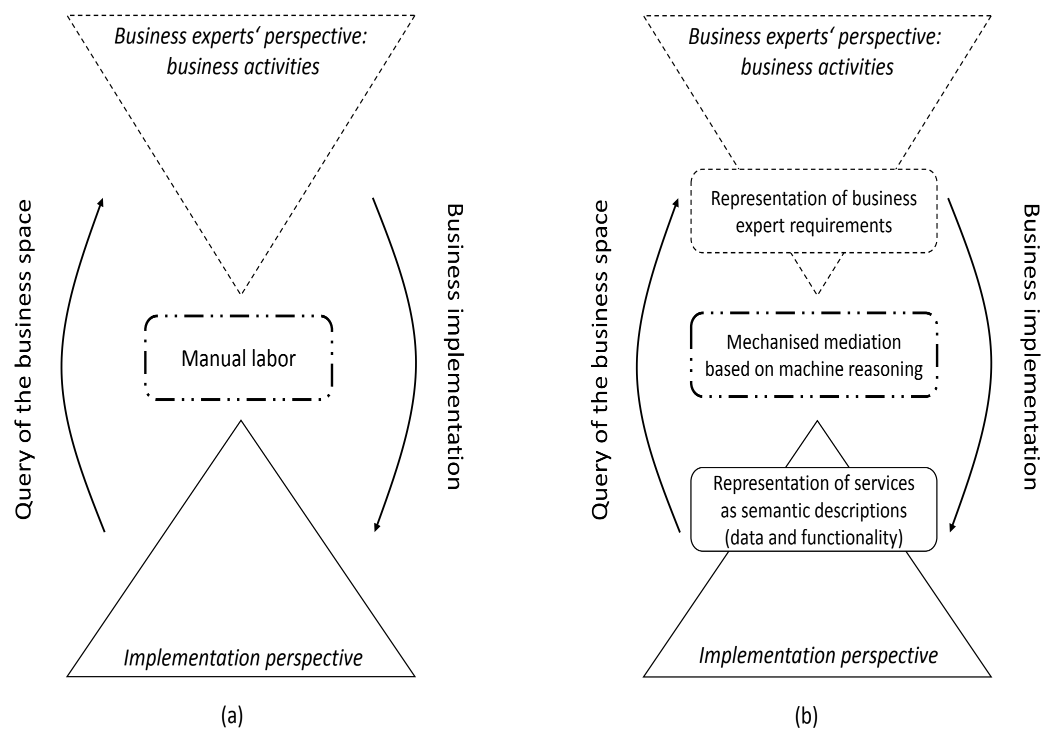

Business experts' perspective: business activities
Query of the business space
Business implementation
Manual labor
Implementation perspective
(a)
Business experts' perspective: business activities
Representation of business expert requirements
Mechanised mediation based on machine reasoning
Representation of services as semantic descriptions (data and functionality)
Query of the business space
Business implementation
Implementation perspective
(b)

Figure 1.2: Business and IT perspective related (a) by manual labor (b) by automated machine reasoning (adapted from (Born *et al.* 07)).

mapping. These mappings can be automatically constructed or adapted through syntactic or semantic matching techniques (Euzenat and Shvaiko 13).

All descriptions are done via *ontologies*. An ontology is a graph-based formal concept and knowledge description system based on formal semantics (Antoniou and van Harmelen 08), i.e., logic. With it, we represent knowledge, such as the above mentioned business processes, but also individual services. The latter pertains to a service's functionality provided, and also its underlying data model. The ontological representations of both, business processes and services, will serve as foundation for the SSC method we analyze in this document.

1.3.2 Artificial Neural Networks

We combine the purely symbol-based ontology approach with a nonlinear, *connectionist* classification approach, an *Artificial Neural Network* (ANN). We concentrate on its capability to encode entities stemming from compositional (e.g., graph-based) structures as distributed representation. A distributed representation is an ANN-internal representation computed by the ensemble of nodes constituting the ANN. Thereby, each entity (fed as input) is represented by a pattern of activity, distributed over many nodes. Furthermore, each node is involved in representing many different entities (Hinton *et al.* 86).

This combination allows for a greater tolerance of changes in the input values than (conventional) symbolic approaches, such as ontologies, to the representation of meaning (Hammer and Hitzler 07, d'Avila Garcez *et al.* 09). Indeed, an ontology provides very precise inferences but without any tolerance (Chan 03). However, we seek this tolerance to achieve the flexibility needed within changing corporate service pools.

Specifically, we feed ontological information to an ANN implementation called *Labelled Recursive Auto-Associative Memory* (LRAAM). With it, the entire arbitrary-sized graph structure of an ontology can potentially be compressed into a fixed-size neural network representation (Sperduti 93, de Gerlachey *et al.* 94). This *reduced* representation is then transformed into a distance measure to quickly compare ontologies, or else, business activities and services.

Our view is that in a highly dynamic environment it is not about finding an exact, but rather a most similar service to support an activity. To support this view, a fine-grained (i.e., continuous) distributed pattern as opposed to a coarse-grained (i.e., discrete) symbolic pattern is evaluated.

1.4 Research questions

Ontologies provide a formalized conceptual view of a domain of interest. It allows for expressing complex types of entities and relations among them (Rittgen 08). Furthermore, ontologies provide a means of conducting automated plausibility checks (i.e., reasoning). They are also known for the potential to become a reliable technique for realizing meaningful data access (Poggi *et al.* 08). In short, ontologies allow for machine readability, encoding and manipulation of domain knowledge, and connectivity.

We consider those aspects appealing to achieve the above described goal which is, again, to save organizations money by means of automating service selection and integration, more than is current practice in the industry. Our primary research question is:

Can ontologies stand-alone, that is, in isolation, be used for an intelligent service selection and integration method?

We are confident to find valuable answers for the following reasons:

- Ontologies represent a powerful base for expressing every-day knowledge (Hurley 03). As mentioned before, we use them to describe business processes, or more specifically, related activities, and applications. The latter includes data model and functionality provided. The description of application data models within ontologies is shown for example in (Trinh *et al.* 06, Xu *et al.* 06, Bagui 09, Sequeda *et al.* 12, Xu *et al.* 12).

- Computer systems can read and "understand" ontologies. It is thus possible to automatically reason over the ontologies' content. The validity of statements can be verified (Baader *et al.* 09), such as whether concepts are non-contradictory or if implied relations among concepts exist. To us, this increases reliability and trust in automatically choosing the right application functionality to support a specific business activity.

- Connectivity. Ontology based data access (ODBA) permits meaningful data access (Kontchakov *et al.* 13, Bagosi *et al.* 14). Thereby, correspondences are automatically selected and transformed into an executable mapping for migrating data from a source to a target schema (Fagin *et al.* 09). Clearly, this capacity goes beyond syntactic matching techniques within the application integration domain[1] where meaning is associated by human experts.

- Different ontologies are based on the same logic language, namely OWL (Ontology Web Language). Consequently, we do not need to care about the semantic heterogeneity of different "grammars", as the OWL primitives' semantics are equal for all ontological representations (Jardim-Goncalves *et al.* 13). We only focus on the semantics encoded within the ontologies themselves. For our research, it pertains to *ontology matching* which is the process of finding correspondences, *mappings*, as a solution to semantic heterogeneity among ontologies (Euzenat and Shvaiko 07).

The last point is of special interest. We describe services and processes by means of OWL-based ontologies. Semantic heterogeneity is inherent, as stated in Section 1.2. Consequently, finding semantic similarity among entities will help to know which service best[2] supports a certain activity. As stated, we assume that such a service's ontological description is similar

[1]cf., Chap. 2.

[2]For a definition of "best", cf., Chap. 2.

to an activity's ontological description. Thus, when ontologies are used for integration the integration problem becomes an ontology matching, that is, a mapping problem.

As briefly mentioned in Section 1.3, from the perspective of changing conditions, however, we think that the above symbol-based mapping alone is not sufficient for an intelligent SSC method[1]. The reason is that ontologies rely on unique naming in order to connect facts to guide some task (DARPA 08). In other words, an ontology based system assumes that a single, well-formed ontology applies throughout the domain of interest.

From Section 1.2, we know that the corporate IT domain does not respect this assumption. The ever changing business conditions lead to a constant need for evaluating the similarity of independent ontologies, either representing business activities, or representing services to support the activities. Automatically establishing mappings, such as based on *precision* and *recall*, respectively *F-Value*[2], on such a large scale however remains a challenge (Diallo 14, Otero-Cerdeira *et al.* 15). As an arbitrary (high) number of ontologies needs to be evaluated continuously, these challenges include *efficiency* in terms of search space and time consumption, *effectiveness* in terms of correct and complete identification of semantic correspondences (Rahm 11), the potential use of possibly fragmented background knowledge, or user involvement (Shvaiko and Euzenat 13), in the present case, beyond organizational borders.

To address some of these challenges, such as correct identification and decreasing the need for human intervention, we complement purely symbol-based mapping methods with a nonlinear classification approach, the LRAAM. With it, we combine the expressiveness of symbolic knowledge representations, i.e., ontologies, with the robustness of ANNs in the face of changing conditions (Hitzler *et al.* 05).

ANN's are made up of a connected network of individual computing elements, mimicking neurons. Often, they are deployed for classification tasks (such as credit-risk assessment), data-processing tasks (adaptive signal processing), or approximation of arbitrary functions (time-series modeling and prediction) (Jones 08). In the context of this thesis, we explore properties such as learning by example and pattern recognition. The secondary research questions we want to answer is:

[1]A number of mapping approaches, all symbol-based, such as terminological, structural, or and formal semantic, are discussed in Section 5.1.

[2]The description of those measures is beyond the scope of this work. Cf., (Rijsbergen 79), (Makhoul *et al.* 99), and (Do and Rahm 02) for further details.

Other than classical symbol-based ontology matching, is an LRAAM, specifically its distributed patterns, a reliable alternative matching technique as part of an intelligent service selection and integration method?

Technically, we are interested in the LRAAM's potential to compress the symbol-based labelled, directed, acyclic graph structure of ontologies of arbitrary size into a fixed-size neural network representation. We want to use the resulting (non-symbolic) distance measure to quickly compare ontologies, or parts of it. (cf., Chap. 6 and 7).

1.5 Contributions

We aim at establishing a technical method to systematically automate the service selection and integration process under changing conditions. It is based on the concepts of ontologies for describing services and processes, and on an ANN, the LRAAM, as matching technique between ontologies. Specifically, we propose an implementation to answer above research questions. Thereby, algorithms for ontology parsing and managing the neural network are executed. We test internal and external validity of the implementation. As a proof of concept, we simulate real-life scenarios, such as changing business activities supported by an adjusting services infrastructure and data flows. Specific contributions of this work are:

- a critical review of current syntax and formal-semantics based application integration and service composition approaches. The context is "automation" vs. "flexibility" (cf., Chap. 2, 3, and 5);
- an encompassing (i.e., global) service selection and composition algorithm (cf., Chap. 4). The algorithm receives and transforms ontologies (process and service representations). It then calls a subsequent ontology matching algorithm;
- an ANN-based ontology matching algorithm (cf., Chap. 6 and 7). It translates the arbitrary-sized compositional structure of ontologies into reduced, fixed-sized vectors (assembled into vector lists). They are used to compare and thus match activities and services; and
- the identification of specific, promising research venues. Those venues address specific phenomena found during the experimentation phase.

1.6 Research methodology

Following (Dodig-Crnkovic 02), computer science research can be categorized in three domains: *Theoretical*, *Experimental*, and *Simulation*. In this research, the adopted methodological process is *Experimental*. We chose it for several reasons: Our experiments might bring unexpected phenomena to light. Based on those, we could potentially correct our approach (Tichy 97). Furthermore, our experiments may lead to new, useful, and unexpected insights, and open new areas of investigation. It seems natural to assume this, if nonlinear classification algorithms, such as ANNs, are involved. They typically exhibit properties not open to deductive analysis (Dodig-Crnkovic 02). Finally, the context of service selection and integration represents a real use case. The use of experiments brings our research closer to that reality.

We conducted the following research steps:

Functional modeling We discuss the existent body of knowledge of information systems integration. It includes syntactic and semantic integration methods (cf., Chap. 2, 3, and 5). Thereby, we focus on standardization/automation and flexibility and identify shortcomings of the current practice (e.g., Sect. 2.5, 3.4, and 5.4). A different service selection and integration approach is derived and introduced. We describe its functionality, advocating a hybrid symbolic/connectionist as opposed to a pure and restrictive symbolic approach to integration (cf., Sect. 4.1, and Chap. 6).

Technical design The functional description is transformed into a technical specification (cf., Sect. 4.2, and Chap. 7). Specific technical aspects are considered and established.

Prototypical implementation The technical specifications are implemented as a prototype, called *OntoProc* (cf., Chap 7 and 8 for further details). Thereby, a number of constraining parameters are defined. It assures a confined and therefore tractable environment to conduct experiments.

Experimentation In *OntoProc*, variables are manipulated and empirical data a generated. Different experiments are conducted and presented (cf., Chap. 8). We discuss and use them to conclusively answer the research questions (cf., Chap. 9).

1.7 Document structure

In Chapter 2, we depict industrial (i.e., syntactic) integration techniques. It includes the enterprise service bus, web services, or service-oriented architectures. We discuss respective shortcomings regarding automation and flexibility. In Chapter 3, we take a closer look at ontologies and ontology-based approaches to integration. Semantic technologies clearly aim at increasing flexibility and automation over syntactic approaches. It is achieved by means of formalized knowledge representation and machine reasoning. In Chapter 4, we derive an ontology-based integration algorithm. It is implemented within a prototype tool called *OntoProc*. The algorithm's goal is to select the best possible chain of services to support a given business process. Our assumption is that business processes and services are represented by independent ontologies. Ontologies can be compared, i.e., matched. Matching pertains to *mapping*. This, in turn, is a central concept of integration, namely mapping of attributes. Chapter 5 presents an overview of classic ontology matching methods. We discuss their merits and shortcomings. We also position them against our approach, which is to use a subsymbolic neural network implementation as matching method. In Chapter 6, we then introduce and explain the LRAAM-based matching method. It is also implemented within *OntoProc*. The respective algorithm is shown in Chapter 7. Thereafter, we explore its capability to provide a matching of ontologies (and consequently a selection of services), superior to the aforementioned symbol-based techniques. To this end, in Chapter 8, we set up and describe experiments we conduct. In light of our research questions, the results are discussed Chapter 9. Chapter 10 concludes the work. We summarize the finding at the general level of information systems integration and draw conclusions for further research venues.

2

Selection and Integration

As stated in Chapter 1, an organization is exposed to a continuous adaptation pressure. Applications and services (subsumed as services) are constantly on- and offboarded with respect to the IS they are part of. Efficient and effective service (or application) selection and integration is of high importance.

When selecting and integrating services under above conditions, however, two opposing concepts need to be reconciled: *standardization* and *flexibility*. The first represents formalization, industrialization and automation leading to warranted efficiency gains, scalability, less-to-no manual intervention, and cost cutting. The second imposes an increasing need for agility to always use the service, which best supports certain business requirements. Thereby, “best support” is defined as follows: With $s \in S$ a service and $r \in R$ a requirement, $\delta : s \times r \rightarrow [0, \infty]$ is a matching function, such that $\delta(s, r)$ represents the level at which service s does not supports requirement r. It follows that if s completely supports r, $\delta(s, r) = 0$, otherwise $\delta(s, r) > 0$. Consequently, a service $s^{\star}$, satisfying the condition $\delta(s^{\star}, r) \leq \delta(s, r)$, $\forall r \in R$, is considered best support for requirements set R.

The tension between standardization and flexibility manifests within “. . . an accumulating resistance against change (Hanseth and Ciborra 07).” As an integrated IS consolidates technically and organisationally, it becomes a *de facto* standard and is therefore increasingly problematic to reverse or change. In line with (Callon 90), the degree of irreversibility depends on the extent, to which (1) it is difficult to come back to a state where the integrated IS was only one amongst many possible choices, and (2) it shapes or forces subsequent service selections and integrations.

In this chapter, we revisit the service selection and integration domains. We discuss the flexibility of standards on an enterprise scale. We look at how quickly services can be selected, connected or disconnected in order to establish, adapt, or reverse IS to follow the changing business domain. Thereby, we are interested in the degree of automation with which selection and integration happens. From a cost perspective, a selection and integration standard which combines *automated / flexible* is clearly preferred over *automated / standardized* or *manual / flexible*, let alone *manual / standardized*.

2.1 Service Selection

From Section 1.3, we know that no commonly accepted way exists for COTS selection (Ruhe 03). After reviewing eighteen specific on-premise service (i.e., application) selection approaches, (Mohamed *et al.* 07) however identified 5 steps which appear in all approaches. The authors refer to these steps as "General COTS Selection (GCS) Process."

- Step 1: Define the evaluation criteria based on stakeholders' requirements and constraints.
- Step 2: Search for COTS products.
- Step 3: Filter the search results based on a set of 'must-have' requirements. This results in defining a short list of most promising COTS candidates.
- Step 4: Evaluate COTS candidates on the short list which are to be evaluated in more detail.
- Step 5: Analyze the evaluation data (i.e. the output of Step 4) and select the COTS product that has the best fitness with the criteria.

The GCS process is geared towards on-premise application selection. Each step represents human involvement. It is time consuming and expensive, not only for the requesting organization, but also for the providers of COTS services. Highly skilled members of the selecting organization or of supporting third-party consultancy offices are engaged in this process. Especially laborious examples of GCS are tender process based on the WTO Government Procurement Agreement (Chen and Whalley 11). To guarantee objectivity, government organizations in main industrialized countries are obliged by this agreement when procuring material or services.

With maturing Cloud-based service offerings, as mentioned in Section 1.1, however, the above GCS, as depicted in 2007, is increasingly challenged. The costs to conduct multi-month to multi-year selection processes becomes just too high to screen the proliferating number of rich[1] Cloud-based service offerings. The above Step 2 alone, which in (Mohamed *et al.* 07) received no further description, transforms into a demanding task itself. Even in more recent research on the topic (Hedman and Andersson 14), this task is not addressed. The authors basically go from a requirements phase directly to a service appraisal phase. Clearly, the assumption is still to have full transparency about relevant potential services to be assessed. They are presumed to be just known through manual research.

Recent Cloud-based service selection methods are depicted in (Klusch and Kapahnke 12), (Sundareswaran *et al.* 12), (Qu *et al.* 13), (Qu and Buyya 14), (Whaiduzzaman *et al.* 14), (Jrad *et al.* 15), or (Modica and Tomarchio 15). Two interesting commonalities can be perceived. First, Cloud-based services are still mainly considered to reside on the infrastructure level, i.e., *IaaS* (Infrastructure as a Service). It follows that service selection focuses on P&A (Performance and Availability) and non-functional aspects, such as price, pricing units, or user feedback. Examples for P&A selection criteria are storage capacity, CPU speed, or memory used (cf., Fig. 2.1). The services' *fitness for purpose*, i.e., what they do, are directly related to the services' *fitness for use*, i.e., how well they do it. If a service consumer requests a certain compute performance (i.e., what), exactly that is (supposedly) delivered (i.e., how well). Because of this simplicity, it makes capturing the service nature for comparison much easier. A certain degree of automation of the selection process can be established through comparison of relevant number- or string-based parameters.

Second, if the services discussed are on the application level ((Klusch and Kapahnke 12), (Modica and Tomarchio 15)), i.e., SaaS, the services are describable in terms of parameter inputs, outputs, preconditions, and effects (IOPEs). IOPE is a functional description depicting the transformation produced by the service. It concerns the required inputs (Input), the generated outputs (Output), the constraints the execution of the service might depend on (Condition), and the effects (Result) the execution has (Fensel *et al.* 11). Example[2]: to complete the sale, a book-selling service requires as input a credit card number and expiration date, but also the precondition that the credit card actually exists and is not overdrawn. The result of the sale

[1]that is, providing similar functionality as on-premise COTS applications, and therefore replacing them more and more.

[2]http://www.daml.org/services/owl-s/1.0/owl-s.html).

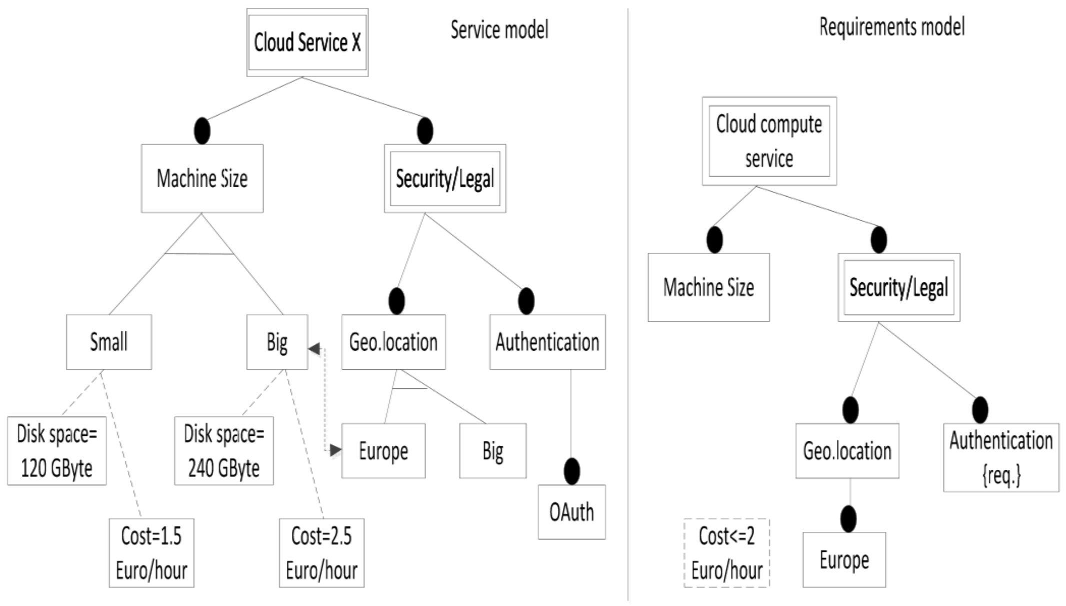


Figure 2.1: Service feature and requirements model: service selection based on few quantifiable criteria.

is the output of a receipt that confirms the proper execution of the transaction, and as effect the transfer of ownership and the physical transfer of the book from the the warehouse of the seller to the address of the buyer. (Klusch and Kapahnke 12) shows that under such a regime powerful automatic selection performance can be achieved. However, they are again based on comparison of string- or number-based parameters.

Therefore, we think that the category of SaaS services, such as provided by SalesForce, ServiceNow, etc. (cf., Sect. 1.1), do not lend themselves to simple IOPE descriptions. They are considerably more complex than the services discussed above, as they provide rich OOTB, yet configurable, functionality, similar to on-premise COTS applications or services.

To us, it is thus necessary to analyze a different way of describing services. Such a description should uniformly apply to services either on-premise or Cloud-based, either simple or rich. Eventually it should serve reliable, automatic comparison to enable the selection of the most adequate service for certain business requirements, to be integrated into the organization's existing service landscape.

In the next section, we further approach the topic as we shed more light on the term *integration*.

2.2 Service Integration

Depending on nowadays sprawling social or technological contexts, integration might have different meanings. Some examples are given as follows:

- In business administration, it is the process of attaining close and seamless coordination between several departments, groups, organizations, systems, etc.[1], or
- in technology, it is a popular buzzword referring to two or more technical components merged together into a single system[2], or
- in information technology, it is the process of linking together different computing systems and software applications to act as a coordinated whole[3], or more specifically,
- in Enterprise Integration (EI), it facilitates the right information at the right time at the right place through connecting all the necessary functions and heterogeneous functional entities (information systems, devices, and people) in order to improve cooperation and coordination so that the enterprise behaves as an integrated whole, therefore enhancing its overall productivity, flexibility, and capacity for management of change (or reactivity) (Vernadat 96), or
- in Enterprise Application Integration (EAI), it is the combination of processes, software, standards, and hardware resulting in the seamless connection of two or more IS allowing them to operate as one (Gupta and Sharma 03).

Based on those descriptions, we consider EAI a subset of EI as it focused on its technical realisation. Moreover, EI and EAI emphasize inner–organizational IS. It relies on a strong decision authority that governs the set up of integrations and IS.

What seems to be missing however is the notion of flexibility. How quickly can a organization adapt existing, or create and put into production, new business processes without IT being too late, too costly, or beside the requirements altogether?

For (Molina *et al.* 98), EI/EAI is therefore rather a vision. It is a life cycle of periodically measuring and closing the gap between changing business requirements and IT support. Clearly, each iteration represents a significant cost factor. It deals with changing technologies,

[1]http://www.businessdictionary.com/definition/integration.html
[2]http://www.webopedia.com/TERM/I/integrated.html
[3]CIS 8020 - Systems Integration, Georgia State University OECD.

and moreover, with people and opinions about what technologies (applications or services) to keep or to replace. Indeed, for the authors of (Lee *et al.* 03), one of the biggest challenges in EI is the impact of experts' beliefs and perceptions. They therefore state that technical integration and "behavioural integration" must equally be taken into account. The latter term – behavioural integration – is noteworthy, because it represents a significant cost driver, as long as a rigorous, reliable, and auto-adaptable, i.e., intelligent service selection and integration support is missing.

2.2.1 Integration for the networked company

One significant development in recent years to address this problem of missing automation, flexibility, and high costs of EI/EAI is *service orientation*.

The basic idea is to modularize and wrap applications behind a formally described access point or interface, e.g., Application Programming Interface (API) . APIs follow rigorous protocols (e.g., SOAP, REST, JSON (Erl *et al.* 14)) and are accessible over the network. Through the API, an application's functionality can automatically be discovered (e.g., using UDDI registries (Kale 14)) and consumed as a *service*. Eventually, the objective is to automatically connect several services together to form meaningful data processing work flows supporting business processes. The architectural principle became known as *Service Oriented Architecture* (SOA) empowering organizations to assemble complex IS with unprecedented flexibility as business requirements shift over time (Erl 04). Following (Kale 14), the essential goal of a SOA is to lower integration hurdles by converting monolithic and static systems into modular and flexible components, that can be reused in many different contexts. For further details, cf., Section 2.4.2.

As stated in Chapter 1, the concept is pushed beyond the corporate limits, for example as a paid third-party Cloud-based services. The latter practically extends SOA into nowadays *Cloud* based service offerings (SaaS).

2.2.2 Loose coupling and interoperability

An organization's goal thus is to become agile and reduce costs. As a Cloud based third-party service is acquired, technical authority over the service and access protocols is lost however. In opposition to organization-internal services, neither of those can be changed nor should it. It could have unintended and unpredictable effects on the usability of the service in other customers' contexts. Therefore, in a service oriented approach, be it internal or Cloud-based, services need to be *loosely coupled*.

Loose coupling refers to services that exchange meaningful data while continuing their own logic of operation (Chen 13). Although this logic involves people, processes, and technologies, none of it is the concern of the data consuming party. In that perspective, classic, i.e., tightly coupling[1], integration does not respond adequately. Indeed, it leads to functional dependencies of components which therefore cannot be separated (Panetto 07). By opposition, loosely coupled services are independent of other services. As mentioned, they can be recombined and reused in different contexts. Again, it makes them more adaptable with lesser costs and quicker implementation (Panetto and Cecil 13) (cf., Table 2.1 for a brief comparison).

Table 2.1: Loose vs. tight coupling (Roshen 09)

Factor	**Loose coupling**	**Tight coupling**
Physical connection	indirect connection through an intermediary	direct connection
Communication style	asynchronous	synchronous
System type	weakly typed	strongly typed
Interaction pattern	distributed logic	centralized logic
Service binding	dynamic binding	static binding

This is especially true for the *networked organization*, an organization that makes intensive use of interactive Cloud-based or on-premise technologies to manage ties with changing external stakeholders – customers and business partners (Bughin and Chui 10). For this organization, integration is therefore less about coupling systems tightly than it is about making them *interoperable*.

Interoperability emphasizes autonomy and flexibility whereas integration refers to coordination difficult to reverse (Chen 13). For (Committee 90), interoperability is the ability of two or more systems or components to exchange information and to use the information that has been exchanged. Following ISO/IEC 2382 (01.01.47), it is the capacity to communicate or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units.

[1]Specific techniques are detailed in Section 2.4.

Interoperability is thus similar to EI/EAI in that different IS operate together to improve business outcomes. It has however a focus on connecting independent applications, or else services, for which no technical authority exists. Tight coupling which entails the possibility of altering IT components themselves to fit an integration is not possible.

In light of our search for flexible and automated on-premise application or Cloud-based service selection and integration standards for dynamic, large-scale IS, we thus focus on integration in the sense of application or service interoperability. Under this angle, IS components, applications and services, are a given to the integration problem rather than an outcome. They evolve over time, but not for integration purposes. They can only be replaced by more appropriate ones. Flexibility to quickly connect or disconnect such components to form adequate IS thus resides within the integration standard.

In the next section, we concretize integration dimensions along which we discuss current integration standards. Thereby, we also situate our specific research which is an intelligent *Service Selection and Composition* (SSC). We focus the discussion on the combination *automated/flexible*.

2.3 Integration dimensions

Based on (Linthicum 99), (Stonebraker 99), (Linthicum 03), and others, (Izza 09) identifies four integration dimensions, namely Scope, Viewpoints, Layers, Levels:

Scope: (Vernadat 02, Singh and Huhns 05) distinguish between *intra-enterprise* and *inter-enterprise* . The authors draw the classical line between integration (within company borders) and interoperability (beyond company borders). As mentioned in Sections 2.1 and 2.2.2, to cope with the rate of change, a company cannot any longer afford to make this distinction. Again, we think that integration and interoperability will need to be the same.

Viewpoints: For (Hasselbring 00, Izza 09), three main viewpoints exist: the *user*'s view (external view), the *designer*'s view (conceptual view), and the *programmer*'s view (internal view). The user's view concerns domain experts and business users. The designer's view concerns the different models used during the design of information systems. The programmer's view refers to the implementation of IS. Our research clearly focuses on the first two viewpoints. However, our ideal of a purely service oriented semantic application selection and integration method would only leave the users' view. It is them who set the business requirements. The designer would be replaced by the automated/flexible system. The developer is irrelevant as the services' functionality is to be known by the system but not its implementation (cf., Sect. 1.3).

Layers: Several integration layers, such as data model, business model, broken down into application interface, method, user interface (Linthicum 99), data and event (Stonebraker 99), or data, message and process (Lubblinsky and Tyomkin 03) have been consolidated into *data*, *message*, and *process* layers. The data layer deals with moving or federating data among multiple data stores. Thereby, data integration bypasses application logic. Following (Izza 09), message or service integration, commonly referred to as EAI, assumes message exchange among applications integrated into IS. Process integration views the enterprise as a set of interrelated processes. It is responsible for handling message flows, implementing rules and defining the overall process execution.

Levels: Four main integration levels, namely *hardware*, *platform*, *syntactical*, *semantic*, are identified (Sheth 99, Xu *et al.* 03). The hardware level encompasses differences in computer hardware, networks, etc. The platform level encompasses differences in operating system, database platform, middleware, etc. The syntactical level encompasses the way the data model and operation signatures are written down. The semantic level encompasses the intended meaning of the concepts in a data schema. For these authors, each level is built on the previous one.

An almost identical view is depicted in (Fenner 03) and (Smirnov *et al.* 03). The authors identify *Platform* as being part of the dimension *Layer*:

- *Platform Integration*: It deals with the underlying physical network architecture but also with the pertaining hardware and management software.
- *Data Integration*: The location of data must be identified, recorded, and a meta data model must be built (a master guide for various data stores). Data can be shared or distributed across database systems, providing it is well-formed and sent through standard formats[1].
- *Application Integration*: The goal is to bring data or a function from one application together with that of another application that together provide near real-time integration.
- *Business Process Integration (BPI)*: It is fundamentally important for a corporation to specify the processes guiding in the exchange of enterprise information. It allows organizations to constantly improve operations, reduce costs, or improve responsiveness

[1]e.g., COM+/DCOM, CORBA, Webservices, JSON, JMS, RMI, XML.

to customer demands. It is a combination of tasks, procedures, organizations, required input and output information, and tools needed for each step in a business process.

For our subsequent discussion, we adhere to the above classifications. We adapt the dimensions *Layer* and *Level* though. Based on (Habermas 84), (Ulrich 01), and (Ludolph 04), we move *Data* to the dimension *Level*. Clearly, in currently realized IS, *Semantic* is not just an abstraction of *Syntactic*, but *Syntactic* is also an abstraction of symbolic *Data*. The dimension *Level* is semiotic in nature. Furthermore, to us, *Hardware* and *Platform* rather belongs to the dimension *Layer*. We see it as a "realization dimension" where *Process* is nothing else than an orchestrated ensemble of application or service components, sending meaningful messages, related through application interfaces thereby forming IS (cf., Sect. 1.1). This classification is summarized in Figure 2.2.

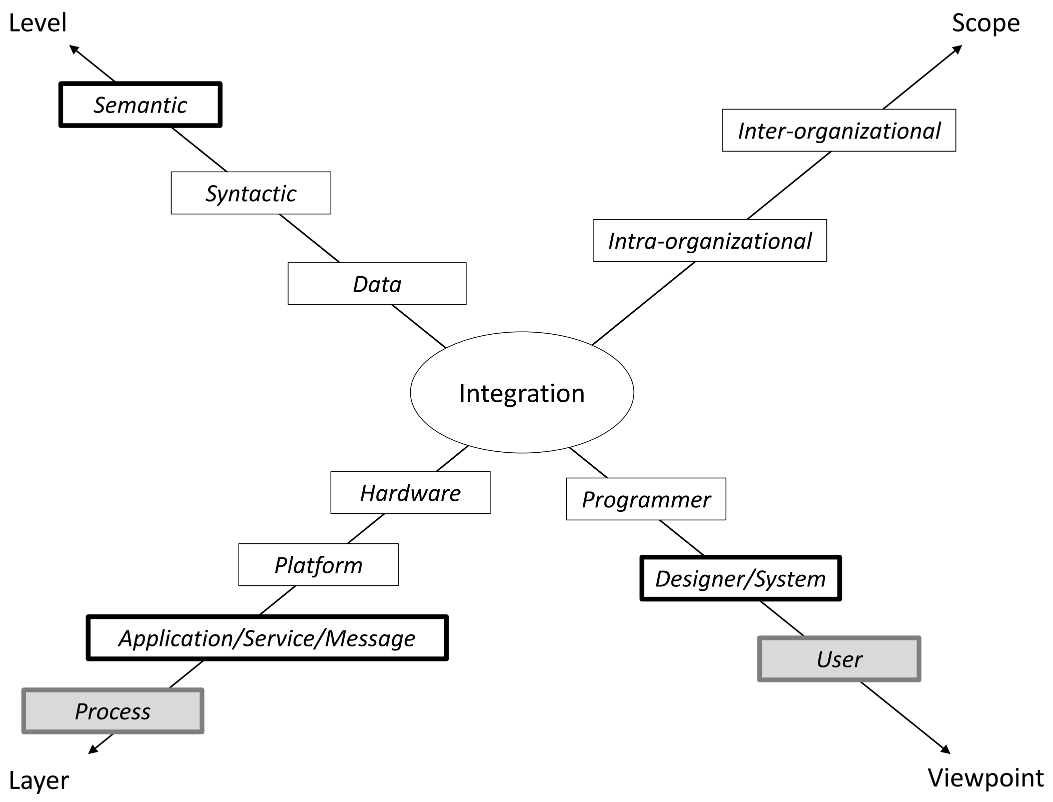


Figure 2.2: Integration dimensions (Izza 09)

The grey box *User* is, as stated before, considered a given. It is outside of the integration approach, but represents the requirements to it. Similarly, the concept *Process* is also situated outside. It is a consequence or outcome of fulfilled requirements as defined by the user. Note, that we consider *Hardware* and *Platform* also as a given and not relevant.

Concluding remarks Based on Figure 2.2, we concentrate on the framed concepts which for us pertain to an intelligent SSC: Through meaningful selection and integration (*Designer / System*), the application components, or else, service (*Applications / Services / Messages*), provide the right data (*Semantic*) to the right place, notwithstanding inter- or intraorganisational, to support a distinctive need, as defined by the user.

Again, the word “meaningful” stands for flexible and automated selection and integration of functionality provided by services/applications. It is needed to cope with changing business activities, which otherwise would bind many financial and human resources.

In the next section, we introduce some current integration techniques. As mentioned in Section 1.3, they are syntactic in nature. For each technique, we discuss its degree of automation and flexibility when connecting services or applications.

2.4 Common integration techniques

Within nowadays IS, *Data* is represented in the concept of linguistic signs, which is also the fundamental unit of database systems (Beynon-Davies 03). Thereby, successful sign communication is the key factor for integration (i.e., interoperability) among services (Liu *et al.* 15).

Following (Saussure 83), a linguistic sign is not a link between a thing and a name, but between a concept, i.e. the *signified*, and a pattern, i.e., the *signifier*. The pattern, e.g., symbol or sound, may be distinguished from the concept associated with it (cf., Fig. 2.3).

In (Chandler 02), three types of signs are distinguished: symbolic, iconic, indexical. In our context, data is of symbolic nature. It pertains to a signifier that does not resemble signified. It is fundamentally arbitrary. Therefore, the relationship between signified and signifier *must be learned*, e.g., alphabetical letters, words, sentences, punctuation marks, or numbers. To quote Charles Peirce, “Nothing is a sign unless it is interpreted as a sign (Peirce 58).” Anything can be a sign as long as someone learns and interprets it as signifying something other than itself (Chandler 02). By extension, the syntax of a data representation specifies a set of rules for combining symbols and arrangements of symbols to form statements in the representation formalism (Beynon-Davies 03).

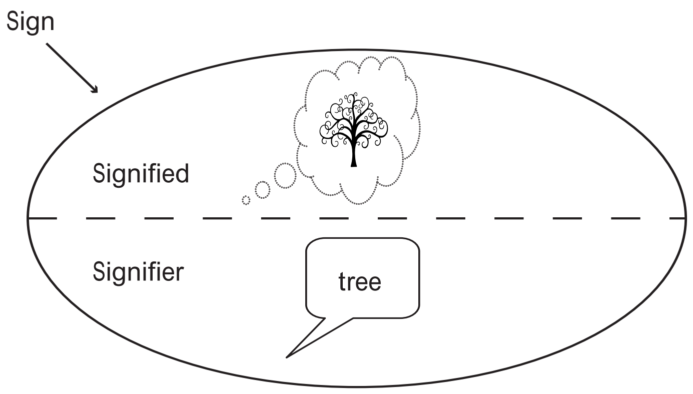


Figure 2.3: Symbolic sign (adapted from (Saussure 83)

Integration techniques in the sense of interoperability thus focus on purposefully moving symbolic data between loosely coupled services and their data stores, which do comply with a specific syntactical standard, but which do not have a meaning in itself (cf., Sect. 3.1 for further details). Prominent examples are the aforementioned SOA (cf., Sect. 2.2.1), which is application-integration centric, or BPI (Business Process Integration), which builds on top of it.

After introducing a simple example we use throughout the next sections, both, SOA and BPI, are discussed in more details.

2.4.1 Running example

Let us assume a simple bank credit loan process as described in Figure 2.4. Grey boxes represent services, which exchange data. They thus need to be integrated. We make no assumptions on the services' delivery model. They may be on-premise, an extra-organizational business partner offering, or Cloud-based provided by a third-party.

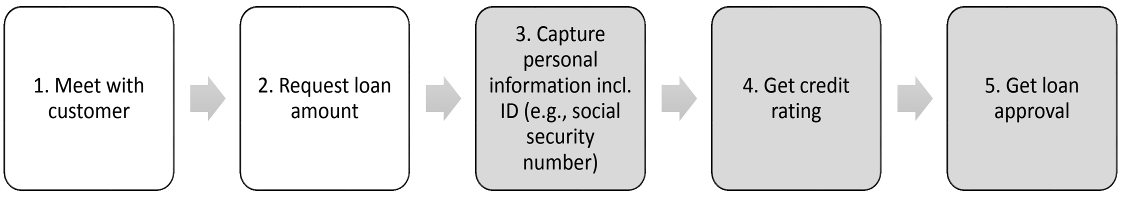


Figure 2.4: Example process. Grey boxes represent services.

Merely, the bank's functional and non-functional requirements[1] decide on the most appropriate service. Here, the question "make or buy" arises. If it is the latter, it might, however, entail a choice among two (or more) services advertised in the market to offer similar functionality.

Let us now suppose that step 4 (Get credit rating) is assumed by such a third-party service. Furthermore, a new service becomes available in the market. It is offered by a competitor. For some reason, it is identical[2] to the old one, except for the rating (or scoring) algorithm it deploys. It is a sophisticated new big-data algorithm, which renders much more reliable scoring values.

We briefly discuss this challenge, namely to automatically exchange the services, for each integration approach presented below.

2.4.2 SOA

SOA (Service Oriented Architecture) focuses on sharing applications' functionality, which is exposed as a service. Programmatically the functionality is accessible through an application's API, as opposed to a user interface. SOA is thus part of the syntactic application integration domain. It emphasizes agile IT systems through reusability of application functionality. Following (Roshen 09), SOA services are not developed to solve business problems. They are generic and simple enough to be assembled in a specific order to meet business needs. This also explains the standardized IOPE-centric reasoning for SaaS-based service selection methods (cf., Sect. 2.1). However, it is in contrast to (Lehaney *et al.* 11), for which SOA services can also represent complicated business processes. For reasons outlined in Section 1, we adhere to the latter. The evolution of SOA is depicted in Figure 2.5 and discussed below.

Sockets

Sockets are communication end points with a name and address in the network. They are created by an application (mostly server) and bound to a port. Then, the application can listen for incoming data on that port. Another application (mostly a client) connects and writes to that socket using the listing application's IP address and the defined port. The listening application reads the data as soon as the other application writes the data.

[1]excluding aspects such as security and privacy, which we assume to be established.
[2]For the sake of the argument, it is not relevant what that implies.

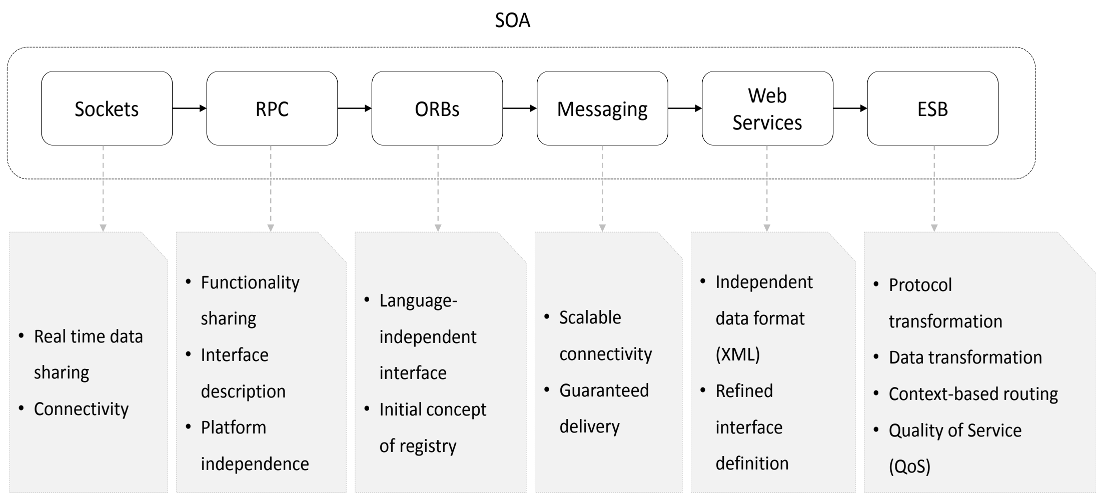


Figure 2.5: SOA evolution (adapted from (Roshen 09)

The connectivity code for this is part of the applications and conceived by programmers. It is deeply entrenched in the application which limits reuse (Roshen 09). It is not possible to share functionality among applications. However, Sockets provide connectivity among application which clearly is the basis for integration, and by extension the aimed intelligent SSC.

RPC

RPC (Remote Procedure Call) builds on top of Sockets. It allows to share functionality among applications (Juric 07). Specifically, an application calls a function of an other application on a remote system (cf., Fig. 2.6). It sends parameters and receives return values.

Through the client stub, RPCs allow access to remote functions as if they were local. However, the integration is programming language dependent and hardcoded "point-to-point," tuned to the call receiving application. It limits automation and flexibility as quick on- and off-boarding of applications, such as needed for flexible IS, is of much effort and assumed by human experts. Furthermore, the roles "caller" (client) and "receiver" (server) are predefined. The client may access the server's functions but not vice versa.

***Running example*:** For business reasons, let us assume the replacement of the credit rating service with a more sophisticated one. RPCs are hard-coded. It would, roughly, entail stopping runtime activities, analysing & renewing RPCs, and testing & moving them to operation to

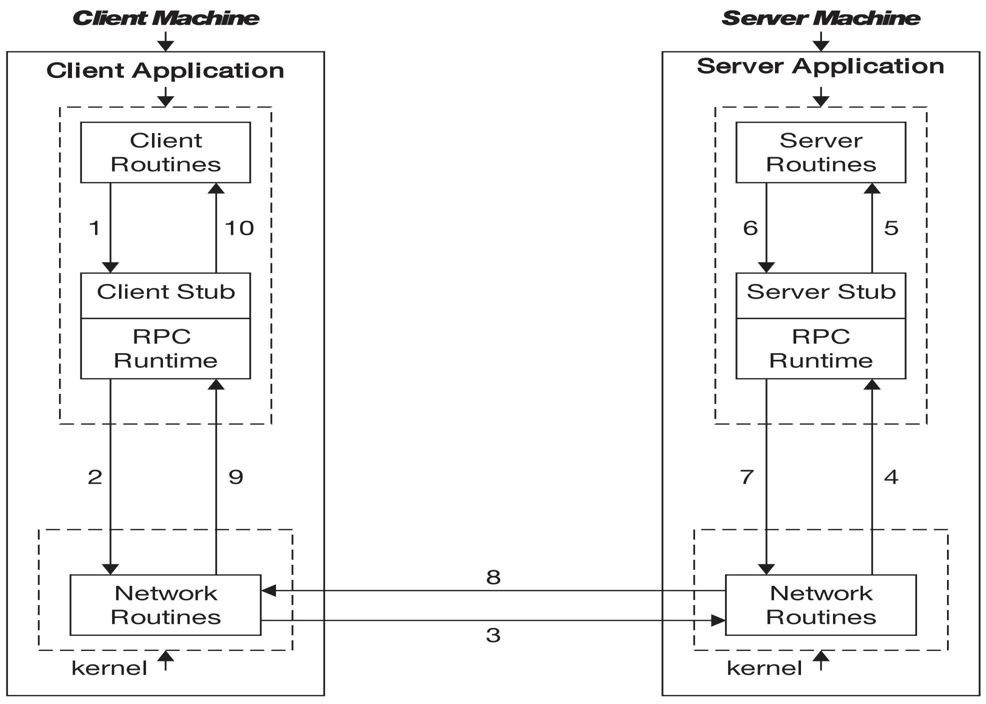


Figure 2.6: Remote procedure call process (Newmarch 12)

resume the process. All tasks are assumed by experts, thus implemented manually. Furthermore, the new service would need to assure the same communication protocol, implementation language, and also a synchronous communication. Eventually, pure RPC limit the replacement activity to in-house services. It represents tight coupling and is not adequate for quick, let alone automatic replacements.

ORB

ORB (Object Request Broker) logically separates an application into objects, which perform specific operations. It also hides network protocol interfaces from the connected applications. It thus enables seamless interoperability between remote objects without the need to look at the connection details (Juric 07). The latter is another shortcoming of RPC, the entanglement of connectivity and marshalling[1] code. Their separation increases usability of the application code. ORB-based integration can thus move away from non-scalable point-to-point integration as the ORB takes care of the connectivity part (cf., Fig. 2.7).

[1]The process of packaging the function call and parameters to be sent in a message (similar to serialization).

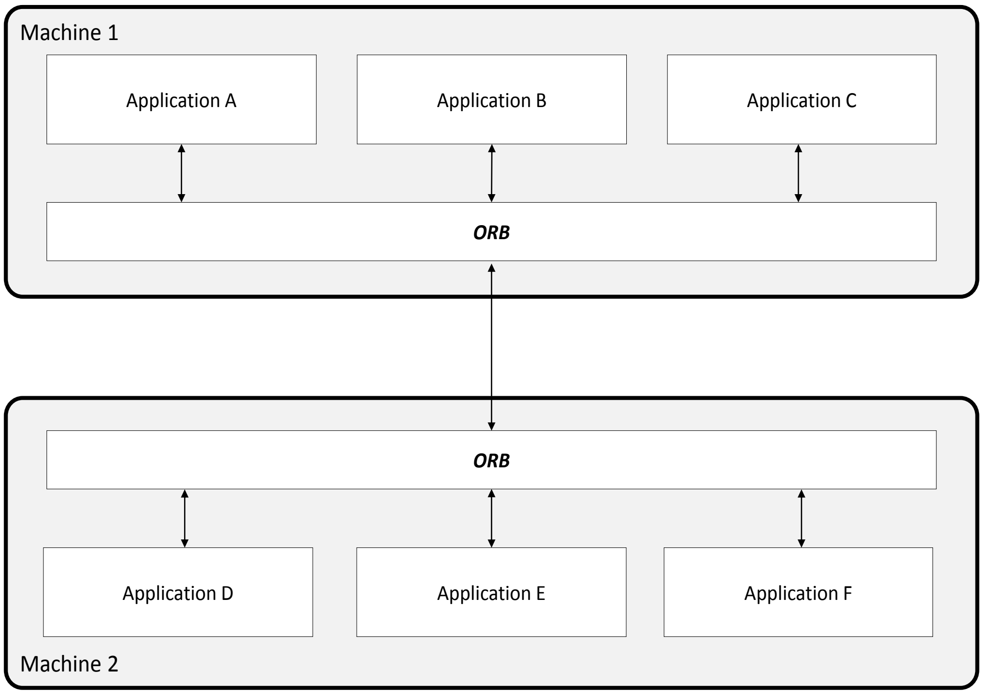


Figure 2.7: General ORB architecture ((Roshen 09))

On transmission, ORB transforms implementation dependent parameters into a platform independent format, IDL (Interface Definition Language). At the receiving end, it is transformed back into the local (maybe different) format. Applications written in different languages, such has C, C++, Java, can now share functionality. This is not possible with RPC.

As implementation boundaries are thus of no concern, flexibility is increased. Applications need to know only what requests they can make and how to make them (i.e., method names and signatures). In principle, adequate composite applications or services can be operated automatically. However, the needed functional integration aspects are again decided and implemented, i.e., coded, by human experts. Down to the symbol, they need to control all ends involved in the integration. Clearly, it poses problems regarding quick adaptation of integrations when reaching beyond organizational boundaries, which is the case with Cloud-based service offerings.

***Running example*:** Similar to RPC, ORB calls are hard-coded. The replacement of the credit rating service with a more sophisticated one would also entail stopping runtime activities, analysing & replacing existing ORBs, and testing & moving them to operation to resume the process. Different to RPCs, the new service could be implemented with a different (object-oriented) programming languages or use a different communication protocol. ORB-based integration also supposes a synchronous communication pattern. Eventually, pure ORB limits the replacement activity to in-house services. It still represents tight coupling and is not adequate for quick, automatic replacements.

Messaging

Messaging is generally an asynchronous method of passing data among applications (Sherif 09). This is different from the aforementioned techniques. Consequently, a temporary storage, a message queue, has to be provided. In it, messages are stored by a sender application until a receiver application connects to it to fetch the data. The message itself is a wrapper of data. It has a header and body. The header is used for control information (i.e., message ID, priority, date, time, origin, destination, etc.), the body contains the payload, such as informative events or commands to invoke functions or methods. The MOM (Message Oriented Middleware) then sends the message to a remote machine where it is read and processed (cf., Fig. 2.8).

One advantage of Messaging over RPC and ORB is that applications do not need to wait for return values, such as the case for synchronous calls. Much larger numbers of applications than with synchronous RPC or ORB can efficiently be integrated as no timing regime is imposed on the applications.

Clearly, the concept leads to a further decoupling of applications. Replacing them quickly is less critical if they communicate asynchronously. It thus provides more flexibility to follow changing business needs. However, operating a messaging system is costly. It requires high maintenance efforts by experts experiencing a steep learning curve (Roshen 09). Once these experts defined and implemented all relevant n-tuples of sending and receiving applications and their specific data needs, it works highly automated and reliable. During operation, the experts also correct payload deficiencies. Syntactic errors, such as truncated strings might otherwise break complex messaging flows. For this reason, Messaging classically presupposes technical and financial authority, limiting it naturally to inner-, possibly extra-organizational integration.

***Running example*:** The replacement of the credit rating service with a more sophisticated one would entail rerouting of request messages to a queue established for the new service. The request and return messages' payload needs to be analysed and possibly changed to suit the

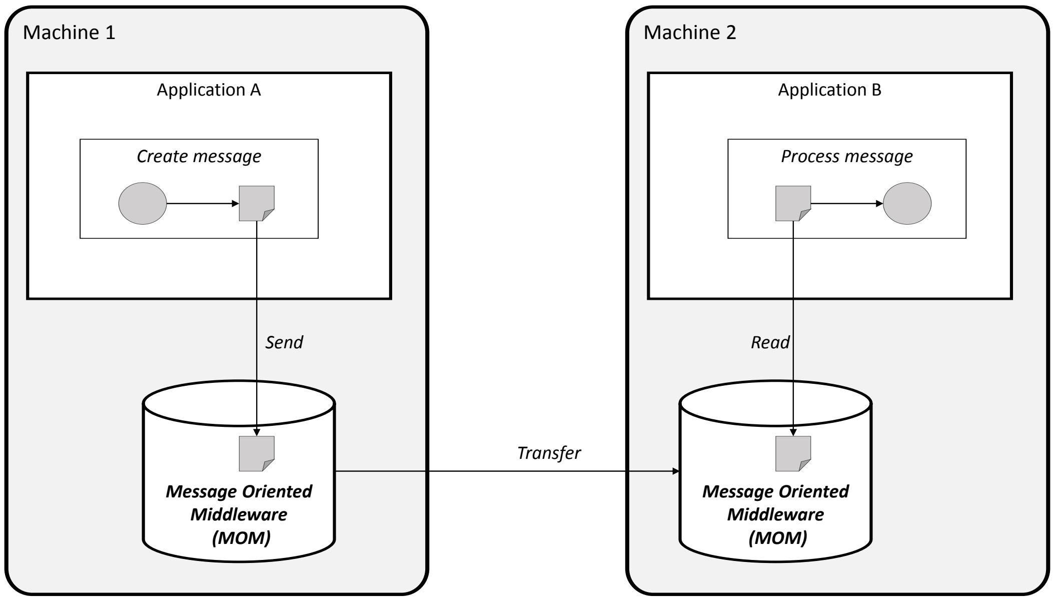


Figure 2.8: Asynchronous message exchange

new service. In general, the looser coupling allows for more flexibility regarding the delivery model. In other words, the new service might come from a business partner without the need for synchronism. However, still some degree of influence over the integration is necessary, as the payload can contain RPCs or ORB-calls. In this case, changes imply intimate knowledge of the new service's inner working. Problems might also arise if messaging systems from different vendors (e.g., IBM, Tibco, Oracle) are used. This is due to a different (incompatible) implementation of relevant integration aspects (e.g., exception handling, fault-tolerance, or administration). Messaging is not flexible enough when Cloud-based services are involved.

Web Services

Web Services (WS) are a crucial part of a SOA. They are self-contained blocks of functionality, i.e., applications, with a well defined interface expressed in a standard format (Ferreira 13). WS are designed to provide services, which are requested and consumed by business processes or other applications being part of the integration. Its general goal is to provide simple access to application functionality for Enterprise Integration (Juric 07).

Its specific goal is *implementation independence*. Integration happens purely on the payload level. It is achieved by means of standard data and data exchange formats, such as XML, SOAP (Simple Object Access Protocol), or WSDL (WS Description Language). This is a further difference to the aforementioned technologies RPC and ORB. As XML is widely accepted, it can also bridge the heterogeneity gap among possibly incompatible messaging systems within and outside an organization (Roshen 09). This aspect is depicted in Figure 2.9.

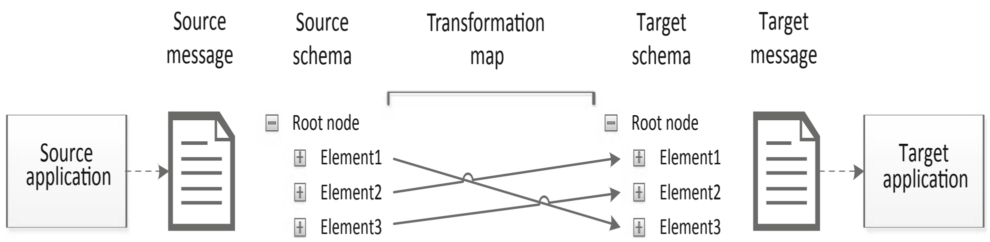


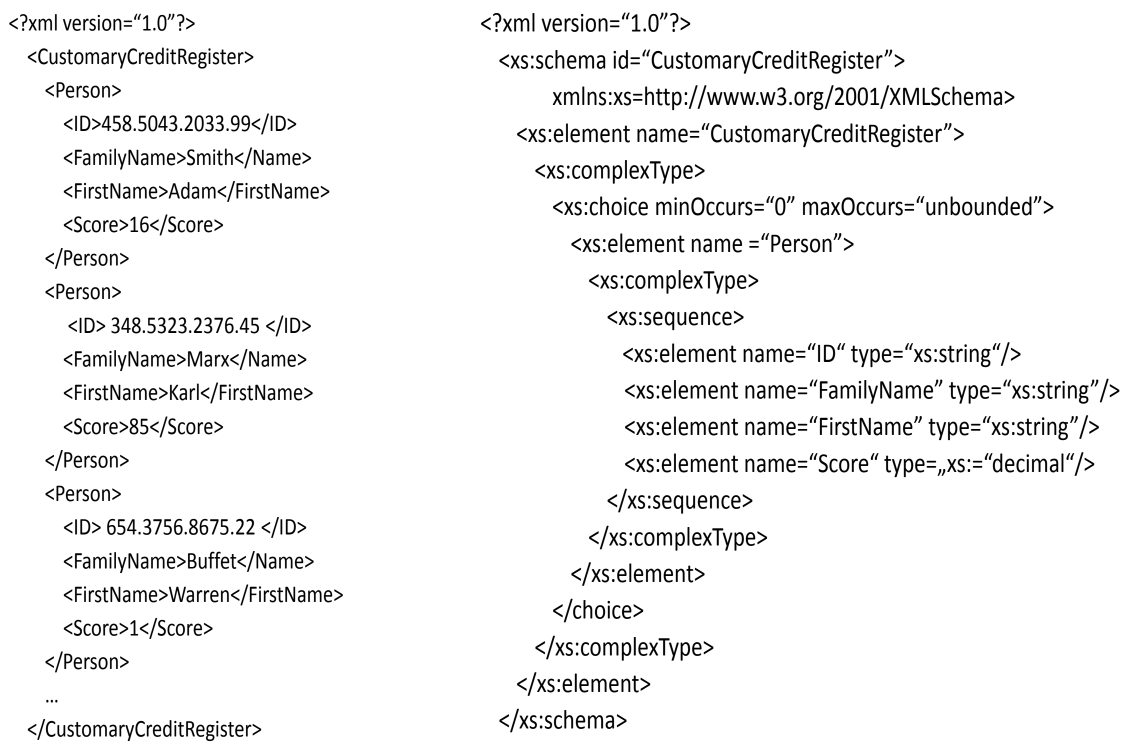
Figure 2.9: Asynchronous message exchange with message transformation (Ferreira 13)

Here, XML-based messages, respectively data within messages, are formally structured within the source schema. Data can be nested. The structure is defined with *XML Schema*. XML Schema defines the grammar, respectively structure of the XML-based data (Kale 14) (cf., Fig. 2.10). Through transformation mechanisms, such as XSLT (Extensible Stylesheet Language Transformation), the source schema can also be transformed into a target schema, i.e., a different XML Schema. The latter may be needed as different applications (or Cloud-based services) need different data structures. Integration, i.e., middleware, tools such as MS BizTalk Server[1] or Pentaho[2] rely on these capabilities. After conceiving the target structure, the target message is generated and sent to the target application for further processing.

WS involve three roles: (1) service provider, exposing the service by means of a network accessible WSDL file, (2) service requester invoking the service through the WSDL file; and optionally (3), a service registry, called UDDI (Universal Description, Discovery, and Integration) where service providers publish information about service capabilities and service interface (Ferreira 13). The latter is needed if the service is not known *per se* but must automatically be detectable. Inner-organizational use of WS however does not normally involve UDDI, as location and nature of WS are known by the experts. It does not justify the costs involved in operating a UDDI registry.

Ideally, however, if WS are defined by the above standards, other services can find and invoke them automatically. More abstract services can be created on top of existing ones.

[1]http://www.microsoft.com/en-us/server-cloud/products/biztalk/
[2]http://www.pentaho.com/product/data-integration



```
<?xml version="1.0"?>
  <CustomaryCreditRegister>
    <Person>
      <ID>458.5043.2033.99</ID>
      <FamilyName>Smith</Name>
      <FirstName>Adam</FirstName>
      <Score>16</Score>
    </Person>
    <Person>
       <ID> 348.5323.2376.45 </ID>
      <FamilyName>Marx</Name>
      <FirstName>Karl</FirstName>
      <Score>85</Score>
    </Person>
    <Person>
      <ID> 654.3756.8675.22 </ID>
      <FamilyName>Buffet</Name>
      <FirstName>Warren</FirstName>
      <Score>1</Score>
    </Person>
    ...
  </CustomaryCreditRegister>
```

```
<?xml version="1.0"?>
  <xs:schema id="CustomaryCreditRegister">
        xmlns:xs=http://www.w3.org/2001/XMLSchema>
    <xs:element name="CustomaryCreditRegister">
      <xs:complexType>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element name ="Person">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="ID" type="xs:string"/>
                <xs:element name="FamilyName" type="xs:string"/>
                <xs:element name="FirstName" type="xs:string"/>
                <xs:element name="Score" type=„xs:="decimal"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </choice>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

Figure 2.10: Sample XML file (left) and corresponding XML schema definition (XSD) file (right)

This ability to aggregate functionality into a higher-level task is called *service composition* (Ferreira 13). Thereby, an important element is the above mentioned WSDL. It is based on XML with a predefined structure. It includes, but is not limited to, XML Schema. Figure 2.11 depicts this structure.

In the <types> section, the data structure is defined. Here, the above mentioned XML Schema elements are used:

```
<wsdl:types>
 <xs:schema elementFormDefault="qualified"
            targetNamespace="http://creditregister.org/">
  <xs:element name="getScoreFromCreditRegister">
   <xs:complexType>
     <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="1"
            name="personID" type="xs:string" />
     </xs:sequence>
```

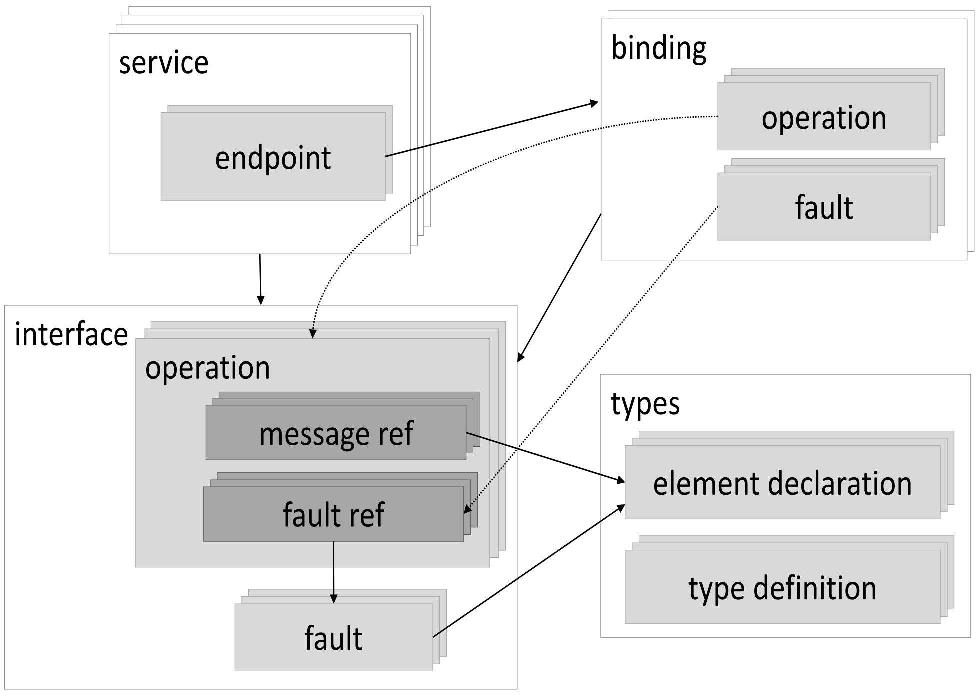


Figure 2.11: WSDL elements (Fensel and Bussler 02)

```
    </xs:complexType>
  </xs:element>
  <xs:element name="scoreResult">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="1" maxOccurs="1"
                name="creditScore" type="xs:decimal"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
 </xs:schema>
</wsdl:types>
```

In the <messages> section, the actual messages, which use the above data structure, are defined. Assumed in the below example is the transmission protocol SOAP. There is one request and one response from the web service:

```
<wsdl:message name="getScoreFromCreditRegisterSoapIn">
```

```
 <wsdl:part name="parameters"
        element="tns:getScoreFromCreditRegister" />
 </wsdl:message>
<wsdl:message name="getScoreFromCreditRegisterSoapOut">
 <wsdl:part name="parameters"
        element="tns:scoreResult" />
</wsdl:message>
```

The <interface> element defines the actual WS. It groups together operations that the WS can perform. One or more of the above defined messages are grouped within an operation. It defines the chronological interaction patterns per operation.

```
<wsdl:interface name="getCreditScore">
 <wsdl:operation name="getScoreFromCreditRegister">
  <wsdl:input message="tns:getScoreFromCreditRegisterSoapIn" />
  <wsdl:output message="tns:getScoreFromCreditRegisterSoapOut" />
 </wsdl:operation>
</wsdl:interface>
```

The <binding> element relates the above defined interface (including operations) with specific transport protocols. Several bindings can exist. For example, one binding might be for SOAP/HTTP with no data encryption for the intranet, and another for SOAP/HTTPS for access from the Internet.

Eventually, the <service> element contains the actual URL (end point) for each binding defined above. The interaction is initiated when a service requester (application) sends appropriately structured requests to those URLs, representing the WS hosting machines. An example of a SOAP request and the response is given below. They are based on above defined WSDL specifications.

```
POST /url HTTP/1.1 Host: www.creditregister.org Content-Type:
application/soap+xml; charset=utf-8 Content-Length: nnn

<?xml version="1.0"?>
 <soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
```

```
  <soap:Body xmlns:m="http://www.creditregister.org/Score">
   <m:getScoreFromCreditRegister>
    <m:personID>458.5043.2033.99</m:personID>
   </m:getScoreFromCreditRegister>
  </soap:Body>
```

In the example, the SOAP request sends a person <ID> to the WS address, which is supposed to return the credit <Score> for that person. It invokes the operation <getScoreFromCreditRegister>. The SOAP reply contains the score.

```
 HTTP/1.1 200 OK Content-Type: application/soap+xml;
charset=utf-8 Content-Length: nnn

<?xml version="1.0"?>
 <soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.creditregister.org/services/Score">
   <m:ScoreResult>
    <m:creditScore>16</m:creditScore>
   </m:ScoreResult>
  </soap:Body>
 </soap:Envelope>
```

As software providers generally support the WS standard, it further increases integration flexibility and automation for organizations. Based on XML, services' access interface are described in a common way, which is intelligible to human experts. Independently of implementation languages and disparate middleware technologies (Chen *et al.* 08), it allows for defining an exchange of complex data structures among applications (Roshen 09). Messages can also be bound to different transport protocols.

Despite those advantages regarding flexibility, WS adhere to the point-to-point integration style. This is different from Messaging. Pure WS integration is thus not scalable as long as it cannot be automated. WS also presuppose equal transport protocols and compatible message structure between requesting and responding application.

***Running example*:** WS rely on XML. The replacement of the credit rating service with a more sophisticated one would entail analysis and possible restructuring of the request message, including data structure and transport protocol. Obviously, this depends on the new service's WSDL file. It defines how to access the service's functionality (through <interface><operation><message><types>). Thereagainst, if both, old and new service have the same interface structure, replacement means only redirecting requests to the new service's end point URL. Integration flexibility clearly increases and allows easy access, even if the WS is Cloud-based.

Enterprise Service Bus

Beside the aformentioned WS, Enterprise Service Bus (ESB) represents the second central part of a SOA. It unites the benefits of both, WS and Messaging. It thereby eliminates the above described WS heterogeneity problems, namely protocol and message format mismatch. The principle setup of an ESB is sketched in Figure 2.12.

Through it, services communicate with each other in a scalable fashion. Application logic accessed through the service interface consumes or provides services without knowing whether the connection is direct or mediated through the ESB (Kale 14). Furthermore, an ESB contains a routing and rules engine with which expert users define and operate the flow of messages, either asynchronously or synchronously. It is shown in Figure 2.13. In a) messages are enriched with external data, in b) messages are split, in c) different messages are merged into one before proceeding.

Clearly, an ESB represents the most versatile integration technique within the realm of current state-of-the-art enterprise application integration. It is less about coding applications so they can "talk" to each other (such as with RPC or ORB) than it is about configuring document structures and exchange patterns (Kale 14). It also transcends proprietary vendor-driven technologies, as organizations demand for standards to cope with a multitude of home-grown and vendor technologies and standards. As (He and Da Xu 14) conludes, ESB and WS, being the cornerstones of SOA, offer a promising framework for flexible enterprise integration.

However, the features of an ESB (e.g., transformation, mediation) are designed and operated by human experts. They implement message processing (i.e., routing, joining, filtering, resequencing, etc.) with a certain degree of flexibility (cf., (Kress *et al.* 13) for further details).

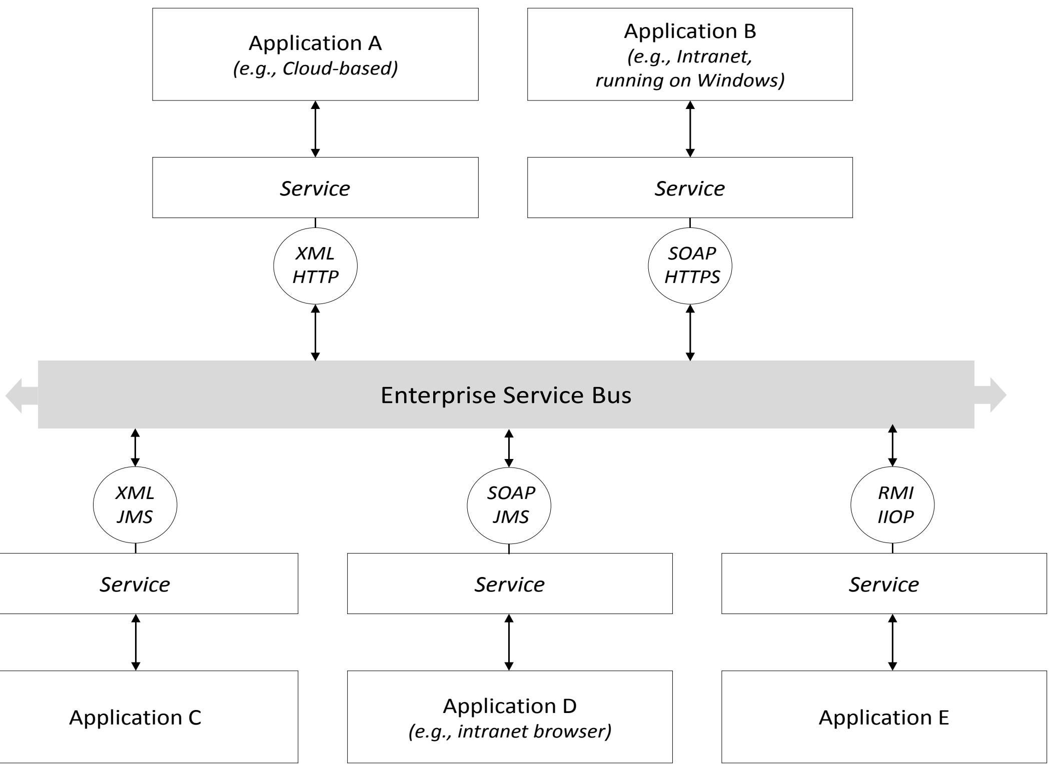


Figure 2.12: Principle ESB architecture, different protocols for transport and data structure can be used (adapted from (Fensel and Bussler 02)

This is achieved by means of simple decision logic based on content, mostly header information. In other words, within predefined limits, the ESB automatically "adapts" its operations. All possible services, end points, and exchange activities have to be foreseen and captured from the start though. This usually happens through event-condition-action rules. Consequently, new services (on-premise or Cloud-based) or new business activities, which may warrant new exchange activities, are not automatically accounted for or incorporated.

***Running example*:** ESB supports WS and transformations & mediation. The replacement of the credit rating service with a more sophisticated one would also entail analysis and possible restructuring of the request message. This time, however, it is done within the ESB. The requesting service does not need to be changed. Flexibility further increases, notwithstanding the service's delivery mode (e.g., Cloud-based). However, automation within the ESB is preconfigured by human experts. There is also a dependency on the ESB as a single point of failure.

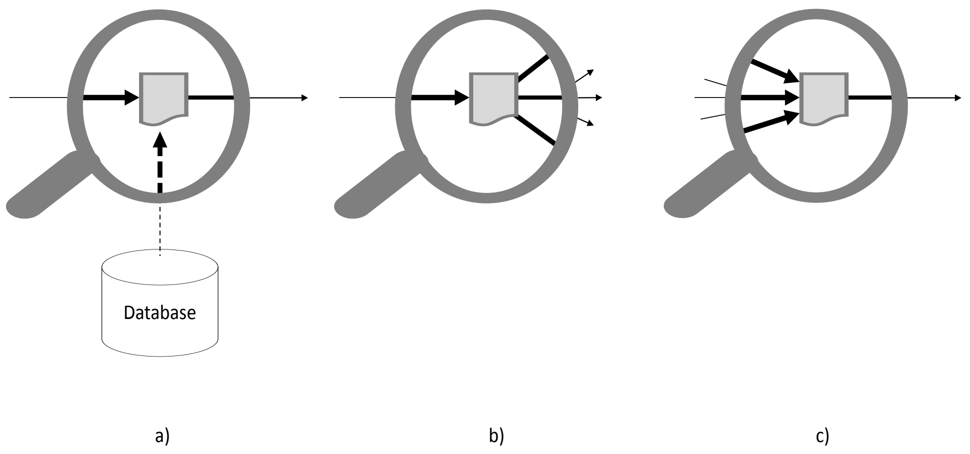


Figure 2.13: Different data handling features of ESBs (Roshen 09)

2.4.3 BPI

The ESB focuses on message processing, e.g., transformations, among services and applications with different data structures or communication protocols. In other words: message in, message out. It is also adequate for handling large volumes of messages exchanges among services (Fasbinder 08). However, those services are generally *stateless* (Kress *et al.* 13). Each service invocation and operation is done as if it were the first one. No memory of its state is retained. An ESB is thus not adequate to orchestrate complex work flows representing business processes (Kress *et al.* 13). As those are oftentimes long running, integration techniques ought to account for the state in which a process is at a certain moment. It must allow for uniquely identifiable process instances. Only then, completion can be assured, or faults can be handled, such as by roll backs to previous states.

It is accomplished by a further layer, which sits on top of XML, WS, and ESB, namely *BPEL* (Business Process Execution Language), depicted in Figure 2.14. Specifically, BPEL (i.e., the BPEL engine) stores the state of each process instance in a database. It assures that execution failures do not cause inconsistencies of the process instance's inner state (Pasley 05). As it provides recovery options for these cases, the underlying WS can remain stateless and light, focusing only on documents and data.

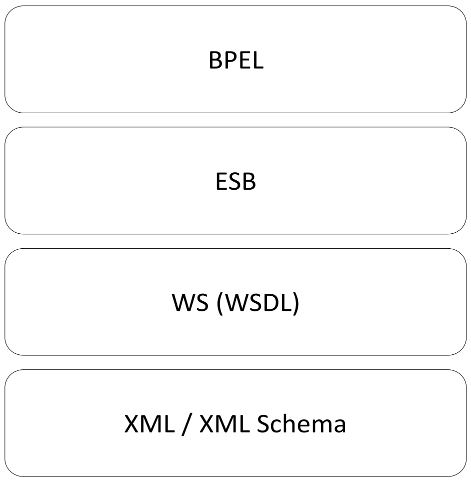


Figure 2.14: SOA-centric BPI (adapted from (Pasley 05), (Silcher *et al.* 12))

Service Composition

BPEL is a programming language, based on XML. It is geared towards business processes (Roshen 09). BPEL guides the concerted operations of multiple Web Services (Cruz-Cunha 09). This is called *Orchestration*. It describes how services interact at the message level. It includes the business logic and execution order of interactions under control of a single end point (Babu and Darsi 13). It uses and extends WS-techniques, such as WSDL, to specify the business process' automatic, stateful execution. From the position of a requesting client, a BPEL - process appears as a standard WS with a predefined <interface>. It is considered a *composed* WS (cf., Fig. 2.15).

Following (Juric 07), through BPEL-enabled WS composition it is possible to modify business processes quickly and therefore provide support to changed requirements faster and with less effort.

BPEL structure

A BPEL process itself is a container where relationships can be declared. It includes external partners, process data, handlers for various situations, and executable actions (Cruz-Cunha 09). The container is represented by the <process> element[1]. It has a name attribute which uniquely identifies the process. It also defines name spaces referred to throughout the declaration.

[1]An exhaustive description of BPEL is outside the scope of this work. Only relevant aspects concerning the couple flexibility/automation are mentioned and discussed. For further details, cf., (Roshen 09).

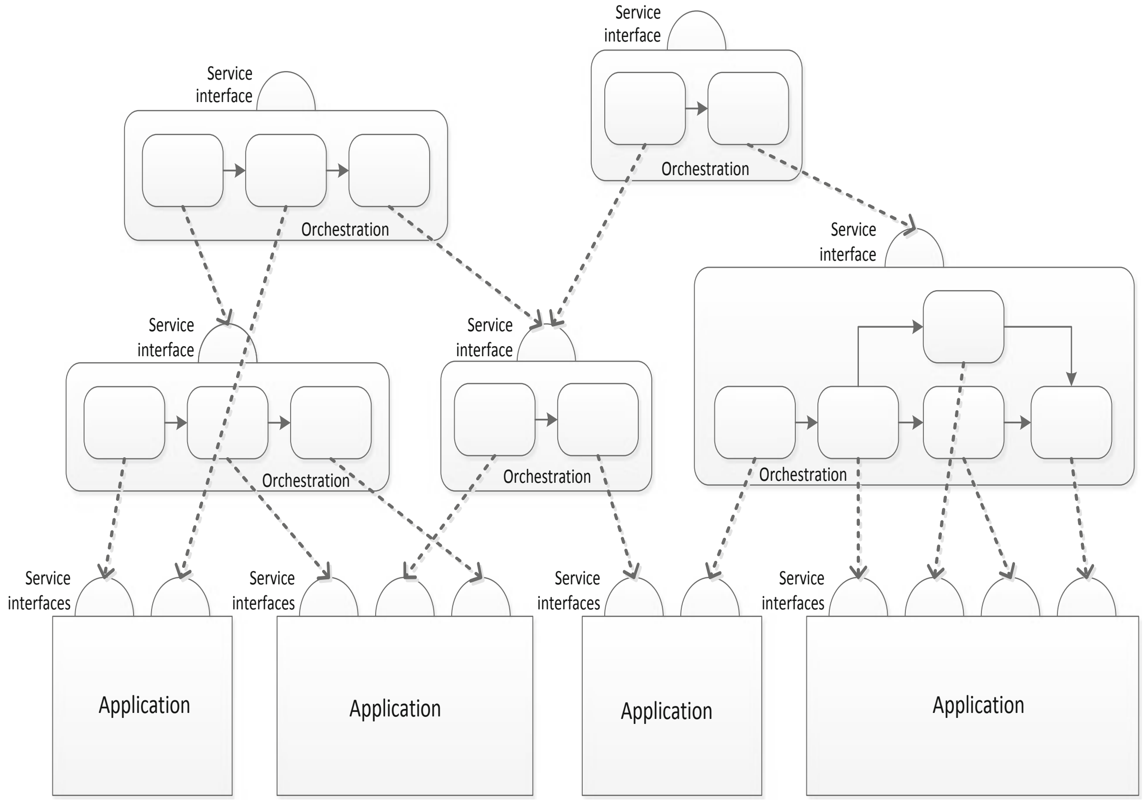


Figure 2.15: Service composition through BPEL (adapted from (Ferreira 13))

```
<process name="loan"
 targetNameSpace=http://example.com/bpel/loan"
 xmlns=
    http://schemas.xmlsoap.org/ws/2003/03/business-process/
 xmlns:bpws=
    http://schemas.xmlsoap.org/ws/2003/03/business-process/
 xmlns:...
```

The <partnerLink> element englobes all WS the process interacts with, and also the process-requesting WS, i.e., the client. Individual WS are specified within the subelement <partner-Links>. The latter also contains information on the specific roles of the respective WS. In the example below, three WS are defined, one requesting client WS and two provider WS, such as for providing third-party information on a person's solvability (score), and for providing the loan approval (for example based on the score, but combined with other criteria).

```
 <partnerLinks>
```

```
  <partnerLink name="client"
     partnerLinkType="bank"
     myRole="bankLoanService"
     partnerRole="bankLoanServiceCustomer"/>
  <partnerLink name="creditRegistrySwitzerland"
     partnerLinkType="creditRegistry"
     myRole="creditRegistryCustomer"
     partnerRole="creditRegistryScoreProvider"/>
  <partnerLink name="loanApprovalBankInternal"
     partnerLinkType="bankInternal"
     myRole="loanApprovalBankInternalCustomer"
     partnerRole="loanApprovalBankInternalProvider"/>
</partnerLinks
```

The <variables> element defines all variables used during the process. Normally, for each message sent to or received from a WS a variable is defined. The declaration is based on XML Schema.

```
<variables>
  <variable name="loanRequest" type="xsd:string"/>
  <variable name="creditScoreRequest" type="xsd:string"/>
      <!-- sends request with person ID (cf., Fig. 2.10) -->
  <variable name="creditScoreResult" type="xsd:decimal"/>
  <variable name="loanApprovalRequest" type="xsd:decimal"/>
  <variable name="loanApprovalResult" type="xsd:boolean"/>
</variables
```

The main part of the BPEL declaration starts with the <sequence> element. It specifies the order in which the partner WS are invoked. Specifically, the first <receive> element is used as initial request from the process' client. The <assign><copy> elements prepare the input for the next WS. The <invoke><receive> elements define the message flow. The last <invoke> element returns the result of the process to the requester WS, namely "client."

```
<sequence>

  <receive partnerLink="client"
         interface="loan"
```

```
       operation="provideLoan"
       variable="loanRequest"
       createInstance="yes" />
<!-- process invocation  -->

<assign>
 <copy>
  <from variable="loanRequest" /> <!-- person ID -->
  <to variable="creditScoreRequest" />
 </copy>
</assign>

<invoke partnerLink="creditRegistrySwitzerland"
        interface="getCreditScore"
        operation="getScoreFromCreditRegister"
        inputVariable="creditScoreRequest"
<!-- requesting score -->

<receive partnerLink="creditRegistrySwitzerland"
        interface="getCreditScoreCallBack"
        operation="getScoreFromCreditRegisterCallBack"
        variable="creditScoreResult"
<!-- receiving score -->

<assign>
 <copy>
  <from variable="creditScoreResult" />
  <to variable="loanApprovalRequest" />
 </copy>
</assign>

<invoke partnerLink="loanApprovalBankInternal"
        interface="loanApprovalBankInternal"
        operation="getApprovalFromBankInternal"
        inputVariable="loanApprovalRequest"
<!-- requesting loan approval, input is the score,
            possibly further enriched with internal data -->

  <receive partnerLink="loanApprovalBankInternal"
          interface="loanApprovalBankInternalCallBack"
          operation="getApprovalFromBankInternalCallBack"
          variable="loanApprovalResult"
  <!-- receiving decision -->

  <invoke partnerLink="client"
    interface="clientCallback"
    operation="clientCallback"
    inputVariable="loanApprovalResult" /
  <!-- return result to client through call back -- >

 </sequence>

</process>
```

BPEL's flexibility

Through BPEL, new processes can quickly be assembled from existing WS and executed. Clearly, it is a more agile method than hard-coding processes (Roshen 09). However, BPEL follows an imperative programming language paradigm (Van Der Aalst *et al.* 03). It focuses on *how* the program operates. It defines sequences of commands for the computer to perform, including the exact definition of parameters, variables, sequence of activities, and constituent WS. In that sense, BPEL offers close control and communication to workflow engines. However, it results in a rigid modeling experience. It can be cumbersome and tedious for the human expert (Tan and Zhou 13). For this reason, (Kapuruge *et al.* 11) argue that BPEL actually still lacks flexibility to adapt to changing business requirements. The authors advocate enhancements to to improve flexibility.

However, (Regev *et al.* 07) state that too much flexibility, that is, numerous modifications, could impair BPEL runtime stability. Clearly, it applies (1) to the executable process itself, and (2) to the constituent WS. First, a process might be impacted by changes to WS, e.g., its data structure. It could break due to subsequent data inconsistencies. After all, WS are self-contained entities, oftentimes beyond technical or functional control of a single organization.

Second, WS might be impacted by changes to the BPEL-process (e.g., invocation load). It could cause WS outages, having repercussion on other processes the WS participates in.

Therefore, (Kapuruge *et al.* 11) proposes to explicitly capture allowable interactions and mutual obligations among the service providers of a composed WS. It is realized by an explicit representation of service relationships. The authors introduce *behavior terms*, grouping relevant interactions and constraints together. Those serve as boundaries for safe modifications and are made available to the BPEL-engine for automatic checks. Following (Kapuruge *et al.* 11), more controlled flexibility can be achieved.

To us, the approach seems valid as long as the set of WS constituting the process is stable. Well-established ties among participating organizations need thus to exist. Otherwise, the administrative overhead, namely negotiating and operating boundaries for each new constituent WS, might outweigh the technical benefits of the approach.

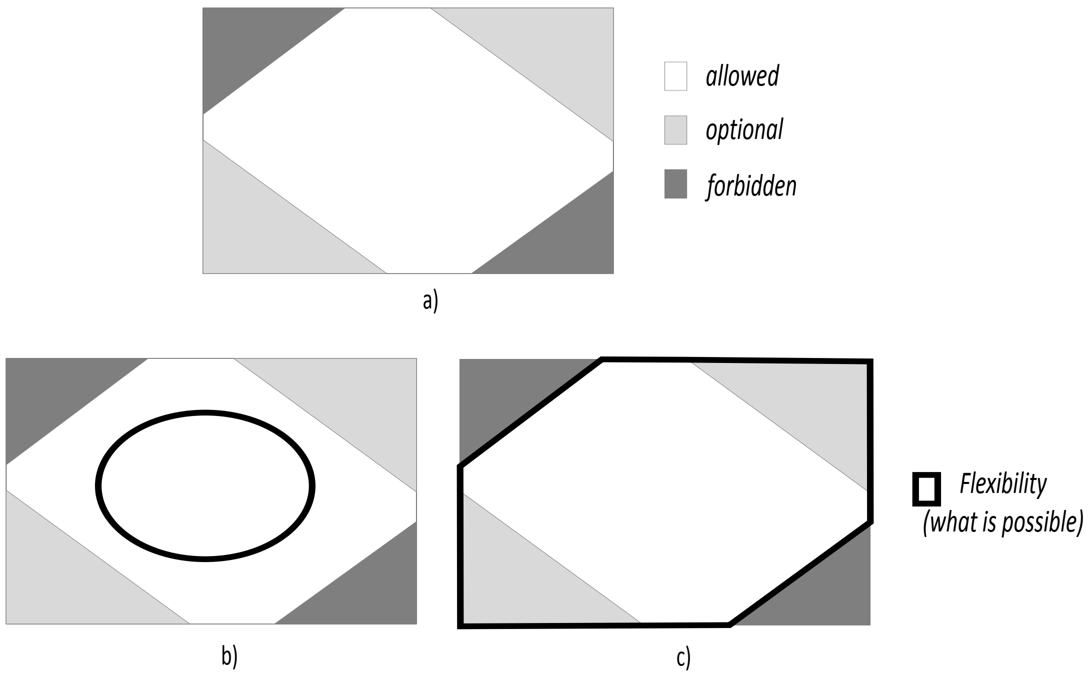


Figure 2.16: Imperative vs. Declarative: a) mandatory and optional constraints b) possible behavior of imperative approaches c) possible behavior of declarative approaches (adapted from (Van Der Aalst *et al.* 03))

In (Van Der Aalst *et al.* 03), a declarative approach to process design is used, which is based on constraints: anything not forbidden is allowed. It is in contrast to BPEL's imperative nature. It is shown in Figure 2.16.

The oval depicted in b) represents the boundaries of BPEL-based process flexibility. Every contingency must be predefined. In other words, all possible execution paths and decision points would need to be incorporated at design time, using an *inside out* style (Van Der Aalst *et al.* 03). This is also true for the aforementioned behavior terms. The frame shown in c) represents an *outside in* style of modeling. Everything that does not violate the mandatory (i.e., forbidden) constraints can be executed. Optional constraints can become violated with a warning message.

The authors claim that it is much more flexible to allow unless explicitly forbidden, instead of forbid unless explicitly allowed. They consider though that the approach may not be suitable for strictly procedural processes. It might be easier to just describe, e.g., with BPEL, what should happen rather than describing the constraints that should be satisfied.

***Running example*:** BPEL allows for service composition. The replacement of the credit rating service with a more sophisticated one would entail analysis and possible restructuring of the composition instructions, namely the BPEL file. It pertains to variables, variable assignments, interface structures, operations, and partnerLinks. From a pure point-to-point integration perspective, BPEL therefore seems less flexible than WS. However, as a *defacto* standard for automatic stateful process execution, BPEL cannot directly be compared to WS, as it is not an integration technique *per se*. However, BPEL also presupposes human experts to assess the structure of new services to be used within a process. With many heterogeneous Cloud-based service offerings to choose from, the process file might either need constant modifications, or more appropriate services are just not used due to high adjustment costs.

2.4.4 SOA & BPI and Cloud

The approaches from (Van Der Aalst *et al.* 03) and (Kapuruge *et al.* 11), depicted in Section 2.4.3, are exemplary and similar to those presented in (Reichert and Rinderle-Ma 06) (design principles for adaptive service flows), (Liu *et al.* 07) (declarative approach to fault tolerance), (Koning *et al.* 09) (extending BPEL with variation points to account for WS variability), (Krizevnik and Juric 12) (automatic data synchronization for long-running processes), or (Domingos *et al.* 13) (global exception handling through process-external context data): To achieve more flexibility, they presume technical authority or, at least, influence over the constituent parts of a process. As mentioned in Section 2.2.2, this premise does not hold for independent Cloud-based services. Let's remember that, to us, services are not adapted to fit integrations. They shall automatically be replaced by more suitable ones based on changing business requirements.

In (Geebelen *et al.* 08), the authors also recognize this problem. They state the independence of third-party services and the possibility of unpredictable behavior, when assembled into business processes. The authors advocate the usage of a generic BPEL master processes and template-based BPEL fragments. The master process describes implementation independent abstract functionality. A template library contains combinations of BPEL activities that fulfill common functionalities. A controller uses both, master process and templates to instantiate executable BPEL processes. To this end, contextual parameters are used, such as the shortest response time. These parameters are delivered to the controller as business requirements (cf., Fig. 2.17).

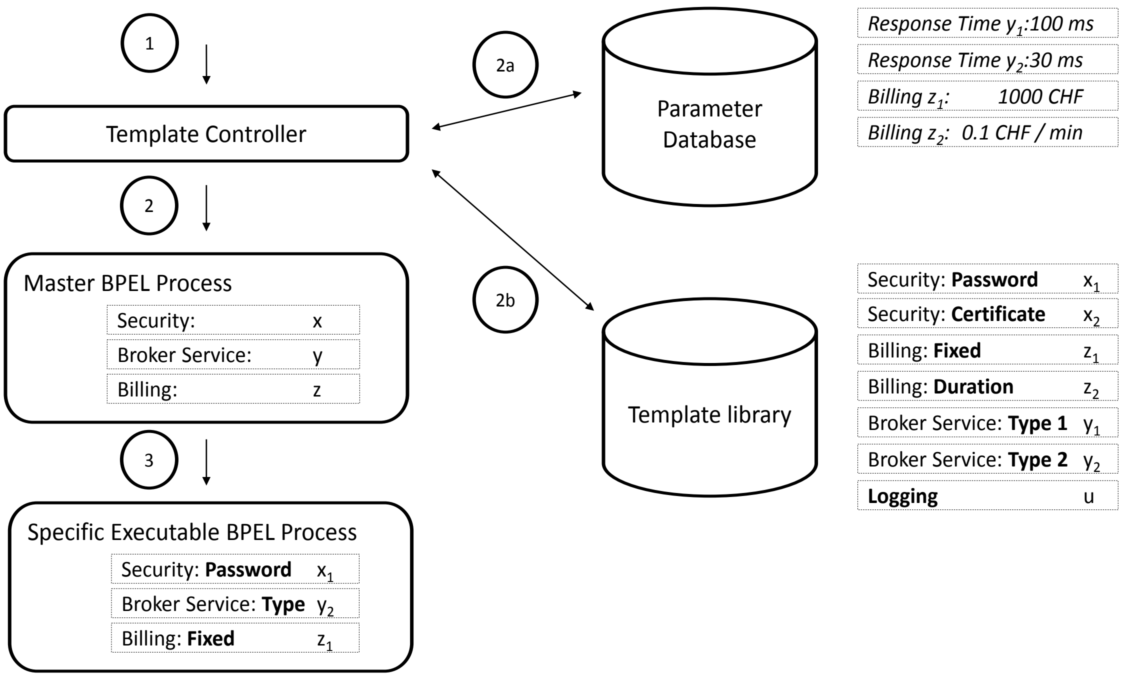

1
Template Controller
2
Master BPEL Process
Security: x
Broker Service: y
Billing: z
3
Specific Executable BPEL Process
Security: **Password** x_1
Broker Service: **Type** y_2
Billing: **Fixed** z_1
2a
Parameter Database
Response Time y_1:100 ms
Response Time y_2:30 ms
Billing z_1: 1000 CHF
Billing z_2: 0.1 CHF / min
2b
Template library
Security: **Password** x_1
Security: **Certificate** x_2
Billing: **Fixed** z_1
Billing: **Duration** z_2
Broker Service: **Type 1** y_1
Broker Service: **Type 2** y_2
Logging u

Figure 2.17: Template framework (Geebelen *et al.* 08)

The authors do not explicitly mention applicability agnostic to the delivery model, i.e., on-premise or Cloud-based. Nevertheless, despite being very exploratory, the idea seems closer to what we seek in terms of automation and flexibility than the sources discussed in Section 2.4.3. A drawback we perceive, however, pertains to the context data for template selection. Those are simple non-functional parameters such as pricing or performance. As mentioned in Section 2.1, those can easily be extracted from simple WS and compared for selection. Already then, we claimed that this is not sufficient for an intelligent SSC as it needs to compare richer information, representing enterprise-scale services, such as SalesForce.

A similar approach is depicted in (Kim *et al.* 12). The authors propose *virtual services* defined as abstracted representation of a service. Where normal WS-BPEL processes need detailed WSDL specifications, virtual services specify a minimal set of information required for identification and assessment. The <virtualInvoke> element indentifies the actual service corresponding to a virtual service (cf., Fig. 2.18).

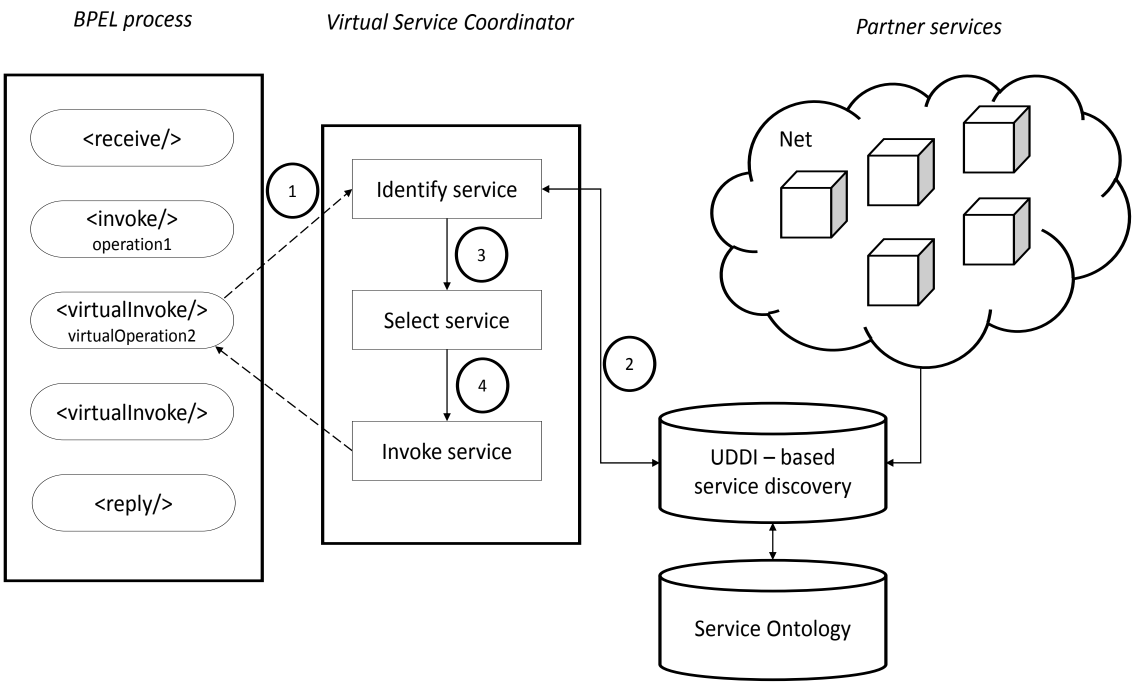
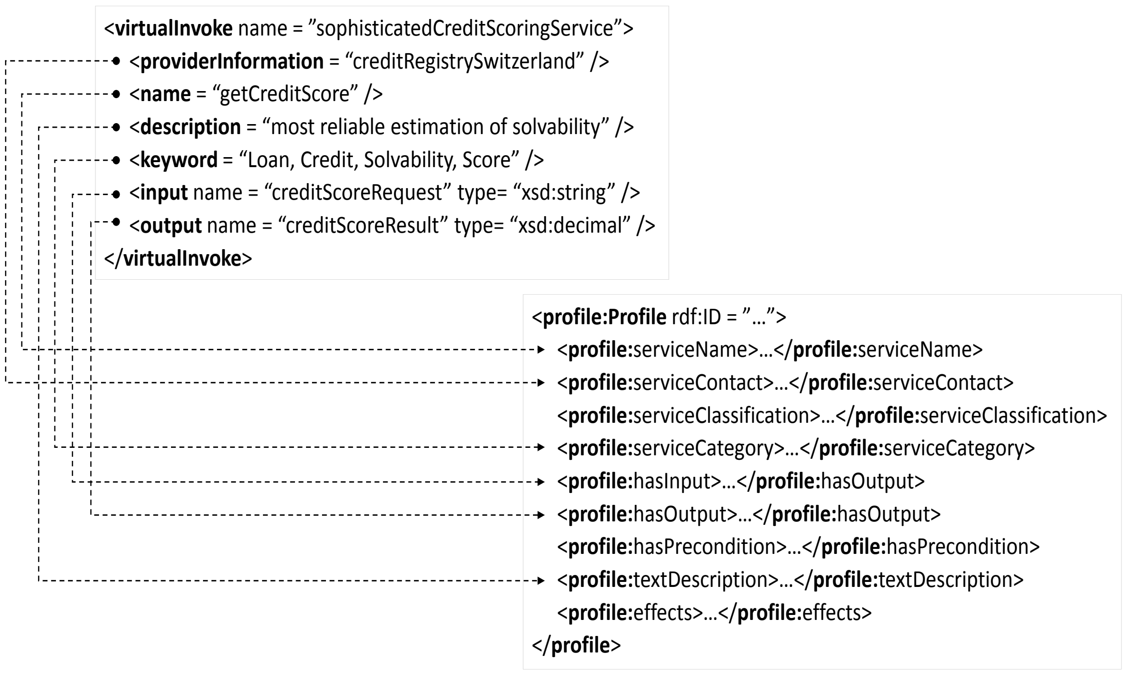


Figure 2.18: Virtual Services Orchestration Framework (Kim *et al.* 12)

The BPEL process is executed. If the <virtualInvoke> element is reached, the coordinator is triggered (1). It identifies potential candidates based on the virtual service's minimal description, namely <providerInformation>, <name>, <description>, <keyword>, <input>, and <output>. The search is performed in a UDDI registry (2). From the identified services, the one with the shortest ping time is selected (3). The service is invoked as part of the BPEL process (4).

Interestingly, the service identification is performed with help of an ontological, that is, semantic description. The authors justify this approach with the UDDI registry's insufficient native search capabilities. Only syntactic keyword matching is provided. They use a semantic service discovery mechanism based on an OWL-S service ontologies[1]. Unfortunately, the authors remain unclear as to how this is achieved beyond what still seems to be simple syntactic matching, such as shown in Figure 2.19.



```
<virtualInvoke name = "sophisticatedCreditScoringService">
  <providerInformation = "creditRegistrySwitzerland" />
  <name = "getCreditScore" />
  <description = "most reliable estimation of solvability" />
  <keyword = "Loan, Credit, Solvability, Score" />
  <input name = "creditScoreRequest" type= "xsd:string" />
  <output name = "creditScoreResult" type= "xsd:decimal" />
</virtualInvoke>

<profile:Profile rdf:ID = "...">
  <profile:serviceName>...</profile:serviceName>
  <profile:serviceContact>...</profile:serviceContact>
  <profile:serviceClassification>...</profile:serviceClassification>
  <profile:serviceCategory>...</profile:serviceCategory>
  <profile:hasInput>...</profile:hasOutput>
  <profile:hasOutput>...</profile:hasOutput>
  <profile:hasPrecondition>...</profile:hasPrecondition>
  <profile:textDescription>...</profile:textDescription>
  <profile:effects>...</profile:effects>
</profile>
```

Figure 2.19: Mapping from Virtual Service to OWL-S (adapted from (Kim *et al.* 12))

The above sources present ideas to render automatic service composition more flexible. The focus is easy reconfiguration of business processes. Thereby, technical control over constituent services is not needed to integrate them. This is in line with our claim that services are a given to the integration problem rather than an outcome (cf., Sect. 2.2.2). Service selection and integration happens by means of "either it matches or it does not." However, the matchings presented are trivial.

***Running example*:** From an automation perspective, the replacement of the credit rating service with a more sophisticated one seems easiest so far in both above described scenarios (with 1: (Geebelen *et al.* 08), 2: (Kim *et al.* 12)). They depict an abstract part, i.e., master process

[1]This is discussed in more details in Section 3.3.3.

(1) or virtual service (2). It is filled with functionality, i.e., templates (1) or specific WS (2). It is based on context data, i.e., response time or minimal service description & ping time (1,2).

Eventually, the systems accomplish the above automatically through some form of comparison, e.g., min(response time) or min(ping time). However, we think that exactly here is the problem. What happens if the more sophisticated credit rating service uses the same signature as the old one? In other words, the requester still sends a personID and is still returned a score. How does the system know that the new service has a much more advanced, i.e., reliable, scoring algorithm implemented? Clearly, a trivial matching, such as based on the above described response or ping time, cannot be the answer. Matching should rather be based on functional aspects.

2.5 Discussing the gap

From the above review, we can summarize that there is wide-spread consensus that flexibility of integration techniques is of utmost importance in nowadays organisations. The evolution of SOA bears witness of this fact. Research in this area condenses around the notion of standardization, service, loose coupling, message exchange, and reusability. XML and derived techniques, such as BPEL, allow organizations to deal with reconfiguring systems rather than constantly having to develop new ones. The ultimate goal is to quickly and automatically support changing business needs.

To a growing extend, this must include exposing services to external organizations. If compared to tight coupling RPC or ORB, it allows for relatively quick setup, but also cutback, of shared processes and enterprise collaboration. However, it still presupposes technical control or at least influence over the services involved (cf., Sect. 2.4.3). Automation therefore is fairly restricted as contingent integration scenarios must be negotiated and implemented. Clearly, it cannot be applied to Cloud-based third-party service (i.e, SaaS) offerings, as they may come (and go) unforeseen.

In Section 2.4.4, ideas are presented to address this issue. Although promising to us, they however use almost trivial selection mechanisms. We claim this to be insufficient in a world with a proliferating number of Cloud-based services. Those will be simple (e.g., weather forecast), but in our point of view also feature-rich (e.g., Salesforce (SalesForce.com 15), Akamaii, ServiceNow).

A convincing case for the latter is made in (Nassif and Capretz 13). The authors rightfully argue that customers cannot chose the features of current Cloud-based, i.e., SaaS-based, enterprise applications they want. They rather have to subscribe to, and pay for, a specific edition, even if certain features are not used. The authors thus suggest that the features of the SaaS applications shall be converted into SOA-based services (cf., Fig. 2.20). They present a 6-stage model to achieve this, including (1) the Determination of SaaS Maturity Level (DSML), (2) Nomination of Application for SOA (NAS), (3) Feature identification, (4) Nomination of features, (5) Service extraction, and (6) Service enrichment.

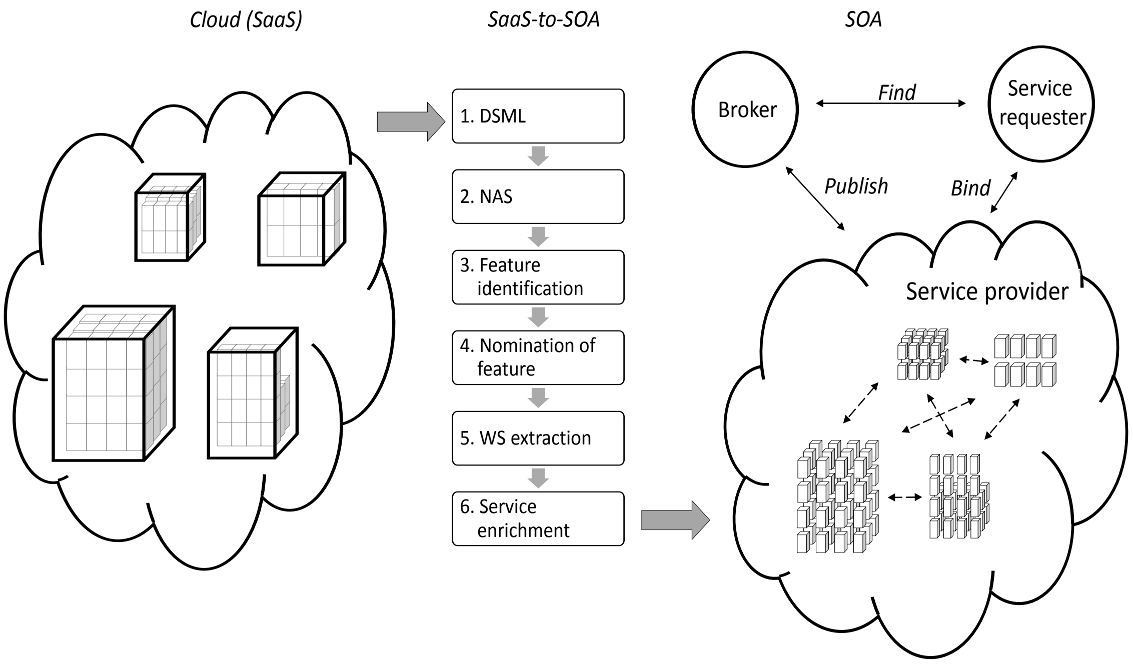


Figure 2.20: Transition from SaaS to SOA (adapted from (Nassif and Capretz 13))

When applied, SaaS providers can publish their services to a broker platform. They could also take the role of a service requester to acquire complementary services from other providers. It may increase the value of own offerings. All other customers acquire specific services from all SaaS providers to fine-tune existing, or even create new, business models of their own.

In such a scenario, one can thus expect to have many competing services with similar functionality because of the *long-term zero-profit equilibrium* phenomenon, described in Section 1.1. Clearly, service selection and integration cannot be based on simple performance values anymore, let alone be done by human experts. It must rely on techniques better capturing a service's functionality. It would include to automatically compare, select, and integrate large amounts of services, based on their description. In the next chapter, we present prominent semantic integration techniques, such as SAWSDL, OWL-S, and WSMO. We discuss their ability to provide these features, which are beyond the above described techniques.

3

Semantic integration

From the prior chapter, we learned that Information Systems (IS) have grown from being intraorganizational and tightly coupled to interorganizational with interactions of multiple systems. Correlated with that evolution towards loosely coupled systems and interoperability is the need for more explicit, machine-interpretable semantics (Obrst 03).

Indeed, Web Service (WS) technologies alone are not sufficient to develop flexible and sophisticated interoperation, i.e., business processes. This is due to the degree of heterogeneity, autonomy, and distribution of Web Services (WS) (Cardoso and Sheth 05). Although they support interoperability between diverse application development platforms, they still require much human interaction (Fensel and Bussler 02, Fensel *et al.* 11). This limits the scalability of SOA-based solutions and greatly cuts the added value envisioned. In order to automate tasks such as reliable WS discovery, selection, and integration, semantic descriptions of WS are thus needed (Fensel *et al.* 11).

Throughout this chapter, we thus explore semantic descriptions in more details. We introduce the concept of ontologies as an indispensable backbone. Thereafter, main specific research venues in the field of semantic integration are presented, such as SAWSDL (Semantic Annotations for WSDL and XML Schema), OWL-S (Web Ontology Language for Web Services), and WSMO (Web Service Modelling Ontology). We discuss the findings in light of our proposed intelligent SSC (Service Selection and Integration). We also come back to our question yet to be answered, namely how the system “knows” that a service is more appropriate than another one.

In light of our research topic, however, we start by discussing the semantics of semantics. We think this to be important in order to better position our research.

3.1 The semantics of *semantics*

Interaction takes place when meanings between entities are exchanged through a common system of signs. The study of those meanings is addressed by semantics (Britannica 15). It deals with linguistic *statements*, either in purely syntactic systems (Formal Semantics) or in natural languages (Cognitive Semantics), e.g., Swiss French (Cherry 67). Formal Semantics pertains to programming languages or standards such as XML. It avoids questions about "meaning" and "truth" with respect to a real world. Cognitive Semantics concerns statements which can only be evaluated and understood under contextual considerations (e.g., psychology or anthropology).

For example, an ontology is an expression of Formal Semantics. It is based on logic which comprises the principles of formal reasoning (Antoniou and van Harmelen 08). Thereby, the meaning of logical statements does not need empirical validation, or the means to do it. The validation process is done by automatic reasoning algorithms (i.e., logical parsers). It ideally takes a finite amount of steps. If it does not, the reasoning process, i.e., the establishing of a statement's validity, is said to be *undecidable*.

Eventually, these aspects can be used to define more sophisticated rules than possible with purely syntactic techniques. Information systems could thus make decisions and select courses of actions. They would follow the principle of $if \rightarrow then$, such as $loyalCustomer(X) \rightarrow discount(X,5\%)$. For simple relationships, inferences are straightforward. In larger sets of statements, automatic inferences become more demanding. It might lead to the impossibility to infer conclusion. Thus, there is a trade-off between computational efficiency and expressiveness (Antoniou and van Harmelen 08).

Cognitive semantics on the other hand centers around *knowing*. Knowing something is to have access to the meaning of what is to be known (Cherry 67). If humans have to process incoming statements, they engage into a process of accessing an existing idea, i.e., the *signified*, that comes to mind in order to be matched against the statement, i.e., the *signifier*. Formal semantic systems (e.g., programming languages, applications) alone are not able to accomplish this assessing task (Harnad 03). They are not able to extract meaning from the statements. It implies that Formal Semantics still only is syntactic manipulation. It does not extend to levels of meaning.

It becomes even clearer when looking at Searle's famous Chinese Room experiment (Searle 80), revisited in (Rodríguez *et al.* 12): In a thought experiment, a person, not knowing

Chinese, manipulates Chinese character strings with syntactic rules written in English. With these rules, the person is able to answer questions asked in Chinese. For people outside the room, the person appears to understand Chinese. However, the person only uses formal features of the manipulated strings. One does not care about the meaning that could be associated with the strings.

For the authors, the argument shows that sophisticated applications, i.e., formal-semantic systems, may appear to be intelligent[1]. In reality though, there is no grasp of the semantic contents of the symbolic structures manipulated.

Clearly, the above notions of Semantics are opposed. Formal Semantics claims to enable machines to understand and, therefore, potentially satisfy user requests, by processing the meaning of data (Fensel *et al.* 11). Through well-defined meaning systems, it can improve current integration technologies. It shall further increase automation and accuracy in various aspects such as information search, extraction, and integration (Berners-Lee *et al.* 01). In this school of thought, machines become thus proactive agents. These agents extract and interpret, rather than just exchange and render information for human users (Fensel *et al.* 11). Thereagainst, Cognitive Semantics states that this cannot be done in a satisfactory way. True knowing and interpreting needs more than just manipulating arbitrary symbolic shapes.

When exploring means towards more intelligent SSC, clearly, these considerations cannot be ignored. During the remainder of this document, our discussion will thus take it into account.

3.2 Semantic knowledge representation

In the following sections, we introduce the concept of ontologies. They are used for formally encoding knowledge. As stated before, ontologies belong to the formal-semantics paradigm. Despite its described shortcomings, in Section 3.3, we will show its advantages for service integration over standard syntactic approaches.

3.2.1 Ontologies

An ontology is a formal, explicit specification of a shared conceptualization (Studer *et al.* 98). A *conceptualization* reflects the semantics of a specific universe's underlying principles

[1]Provided that a conversation in natural language is considered intelligent.

(Guarino *et al.* 09). Following (Genesereth and Nilsson 87), it is an abstract, simplified view of the world[1], henceforward called *domain*, that someone wishes to represent for some purpose.

Specifically, an ontology consists of (atomic or non-atomic) statements and relationships among them. The statements denote concepts. These are also called classes, which belong to the domain, and relationships, which typically express hierarchies of classes (Antoniou and van Harmelen 08). Ontologies also include other statements, such as properties ($x\ teaches\ y$, where $teaches$ corresponds to a binary predicate), value restrictions (only faculty members may teach courses), disjointness statements (faculty and general staff are disjoint), specifications of logical relationships, between objects (every department must include at least ten faculty members).

The notion *explicit* refers to an explicit definition of the classes, relationships and other above mentioned statements. *Formal* implies that an ontology shall be machine readable. *Shared* says that an ontology captures consensual knowledge (Niwattanakul *et al.* 07). It is agreed-upon by a group, having an interest in formalizing a common understanding, e.g., companies within the same supply chain.

Figure 3.1 shows an example ontology. It is aligned to the running example from Section 2.4.1. It sketches elements pertaining, for example, to a financial service provider's offering. The elements, which describe a service named *CreditChecking* are bold-framed. An extraction of the XML-based ontology file is presented below.

```
<Ontology xmlns="http://www.w3.org/2002/07/owl#"

    ...

    <Declaration>
        <Class IRI="#CalculateInsolvencyRisk"/>
    </Declaration>

    ...

    <Declaration>
        <ObjectProperty IRI="#uses"/>
```

[1]Note the semantic synonymity of *universe* and *world*.

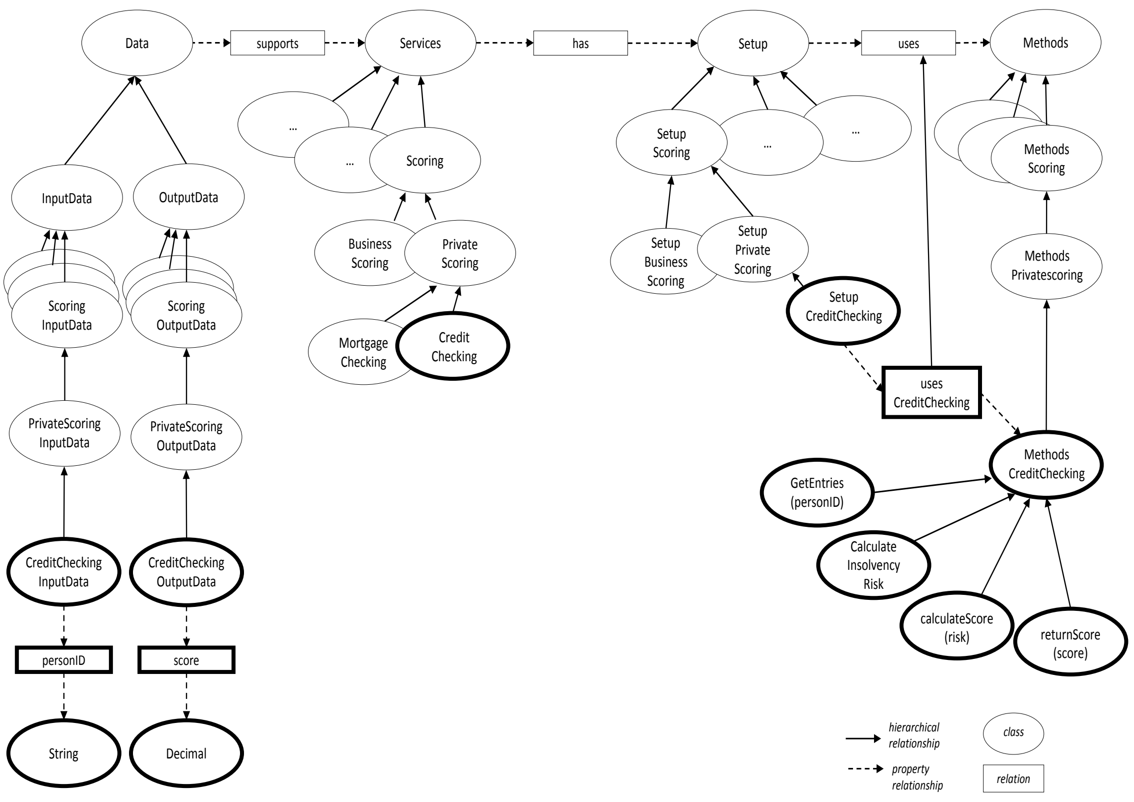


Figure 3.1: Example ontology (derived from running example)

```
</Declaration>

...

<Declaration>
    <DataProperty IRI="#personID"/>
</Declaration>

...

<SubClassOf>
    <Class IRI="#SetupPrivateScoring"/>
    <Class IRI="#SetupScoring"/>
</SubClassOf>
```

```
    ...

    <SubObjectPropertyOf>
        <ObjectProperty IRI="#usesCreditChecking"/>
        <ObjectProperty IRI="#uses"/>
    </SubObjectPropertyOf>


    ...

    <ObjectPropertyDomain>
        <ObjectProperty IRI="#has"/>
        <Class IRI="#Services"/>
    </ObjectPropertyDomain>

    ...

    <DataPropertyDomain>
        <DataProperty IRI="#personID"/>
        <Class IRI="#CreditCheckingInputData"/>
    </DataPropertyDomain>

    ...

    <DataPropertyRange>
        <DataProperty IRI="#personID"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataPropertyRange>

</Ontology>
```

3.2.2 Ontological languages

The above presented ontology is conceived with an ontological language. It allows users to formulate the explicit, formal conceptualizations of the domain of interest. The main requirements are (1) a well-defined syntax, and (2) a well-defined semantics allowing sufficient expressive power and efficient reasoning support (Antoniou and van Harmelen 09). *Well-defined syntax* is a necessary conditions for machine-processing, similar to programming languages. *Well-defined (formal) semantics* refers to subject-independent interpretations. It relies on predefined inference rules established by logics. As mentioned before, formal interpretations are not subject to different points of view of different persons. They allow for automatic and tractable *reasoning*, for example about class membership, equivalence of classes, consistency, or classification. Such inferences can be made mechanically. This is the great advantage over pure syntactic systems, such as described in Section 2.4.

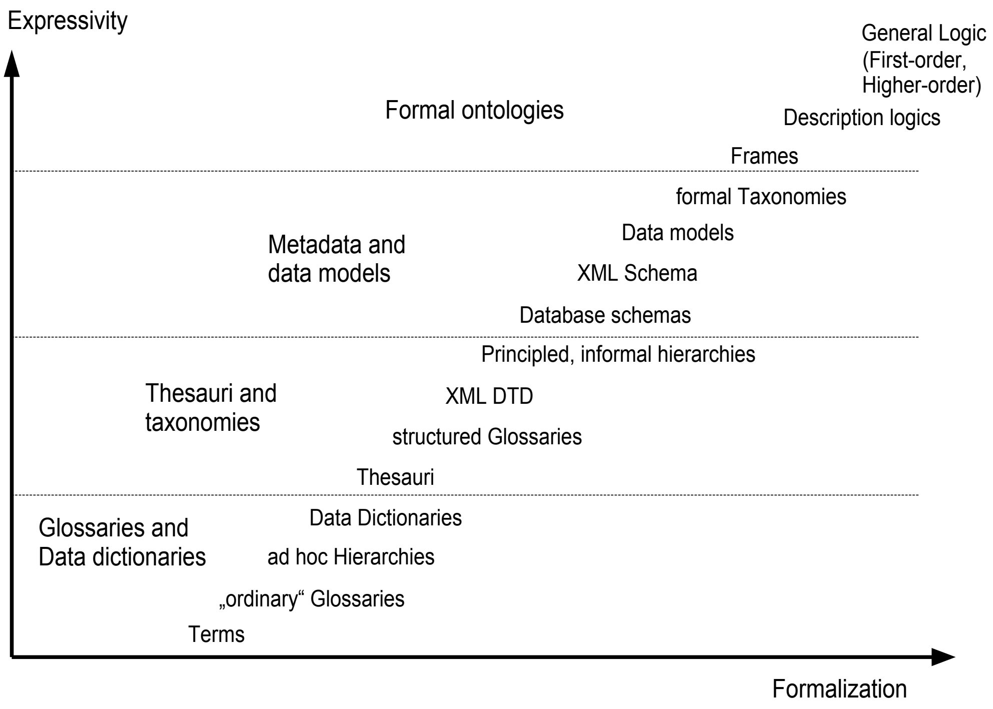


Figure 3.2: Syntactic and semantic data description formalisms (Rebstock *et al.* 08)

The formal semantics and reasoning is usually provided by mapping an ontology language to a logical formalism (Antoniou and van Harmelen 09) (cf., Fig. 3.2, e.g., Description Logic). However, the efficiency of automated reasoning depends on the richness of that formalism. If it is unbound, such as for full first-order predicate logic[1], it becomes too rich. The reasoning task might thus lead to undecidability or at least to heavy computational costs.

The next subsection introduces *OWL* (Web Ontology Language), divided in OWL Full, OWL DL (Description Logics), OWL Lite. They form a family of promising languages to

[1]The topic of logic is not within scope of this work.

balance this trade-off. Each is mapped to a different logical formalism.

OWL Full

OWL Full and OWL DL (Allemang and Hendler 08) use all language primitives provided by OWL. However, OWL Full allows to combine these primitives in arbitrary ways. For example, primitives' pre-defined meanings can be changed by applying the primitives to each other. For example, in OWL Full, a cardinality constraint could be imposed on the class of all classes. This limits the number of classes that can be described in any ontology. Thus, OWL Full offers meta-modeling capabilities, but at the expense of computational tractability (Antoniou and van Harmelen 09). The language is so powerful as to be undecidable, likely disallowing complete and efficient reasoning (Motik 05).

Main language primitives are:

- Class elements
  - Class: *owl:Class*
  - Disjointness: *owl:disjointWith*
  - Equivalence: *owl:equivalentClass*
  - Most general *class: owl:Thing*
  - Empty class: *owl:Nothing*
- Property elements (predicates)
  - Object properties: *owl:ObjectProperty* (relates objects to objects)
  - Data type properties: *owl:DatatypeProperty* (relates objects to data type values)
  - Inverse properties: *owl:inverseOf* (*isThaughtBy - teaches*)
  - Equivalent properties: *owl:equivalentProperty* (*lectures in - teaches*)
- Property restrictions
  - Sub classes: *rdfs:subClassOf*, *owl:Restriction*, *owl:onProperty*, *owl:allValuesFrom* (universal quantification) / *owl:hasValue* (singular proposition) / *owl:someValuesFrom* (existential quantification)
  - Minimum cardinality: *owl:minCardinality*

  - Maximum cardinality: *owl:maxCardinality*

- Special properties:
  - Transitive property: *owl:TransitiveProperty* (*is taller than*)
  - Symmetric property: *owl:SymmetricProperty* (*has same height as*)
  - Functional property: *owl:FunctionalProperty* (exactly one unique value for an object, e.g., *age*)
  - Inverse functional property: *owl:InverseFunctionalProperty* (uniqueness, no two objects have the same value, e.g., *isEmployeeNumberOf*)
- Boolean combinations of classes
  - Complementarity: *owl:complementOf* (same as *owl:disjointWith*)
  - Union: *owl:unionOf*
  - Intersection: *owl:intersectionOf*
- Enumerations
  - Element lists: *owl:oneOf* (defined on classes, e.g., *daysOfWeek*)
- Instances
  - Class instance:
    - *<rdf:Description rdf:ID=“Bob“>*
    - *<rdf:type rdf:resource=“#academicStaffMember“/>*
    - *</rdf:Description>*
  - Different specific individuals: *owl:differentFrom*
  - Pairwise difference of many individuals: *owl:AllDifferent* in concert with *owl:distinctMembers*

OWL DL

OWL DL is the most expressive decidable sub-language of OWL (Pan 09). It reestablishes computational efficiency by restricting the usage of the language primitives. For example, it is not allowed to apply them to each other to alter their intended meaning. Furthermore, any resource is allowed to be only either a class, a datatype, a datatype property, an object property, an individual, a data value, or part of the built-in vocabulary (Antoniou and van Harmelen 08). It follows that a class cannot be an individual, a property cannot have some values from a datatype and some values from a class. This restriction is called *vocabulary partitioning*. Furthermore, the partitioning must be stated explicitly. It is not allowed to have implicitly entailed classes as would be possible within OWL Full. The restriction is called *explicit typing*. *Property separation* says that inverse properties, and functional, inverse functional, and symmetric characteristics can never be specified for datatype properties (Antoniou and van Harmelen 09). Moreover, no cardinality restrictions may be placed on transitive properties. Finally, anonymous classes are only allowed in the domain and range of $\mathrm{owl:equivalentClass}$ and $\mathrm{owl:disjointWith}$, and in the range (not the domain) of $\mathrm{rdfs:subClassOf}$.

OWL Lite

Within OWL Lite, most restrictions apply (Antoniou and van Harmelen 09). For example, it excludes enumerated classes, disjointness, union, complement, hasValue statements, and arbitrary cardinality. Statements about equivalent classes cannot be made on anonymous classes, but only between class identifiers. It is the easiest language to grasp and implement but has the poorest expressiveness.

3.2.3 Ontology types

Ontologies are typically classified on their level of generality, respectively scope (Guarino 98, Roussey *et al.* 11, Prestes *et al.* 13). This is depicted in Figure 3.3. The scope of a local ontology is narrower than the one of a domain ontology. The latter has more specific concepts than a core reference ontology, which is based on fundamental concepts of a domain. Top level (foundational) ontologies describe very general concepts like space, time, matter, object, event, action, etc. They are independent of a particular problem or domain (Guarino 98).

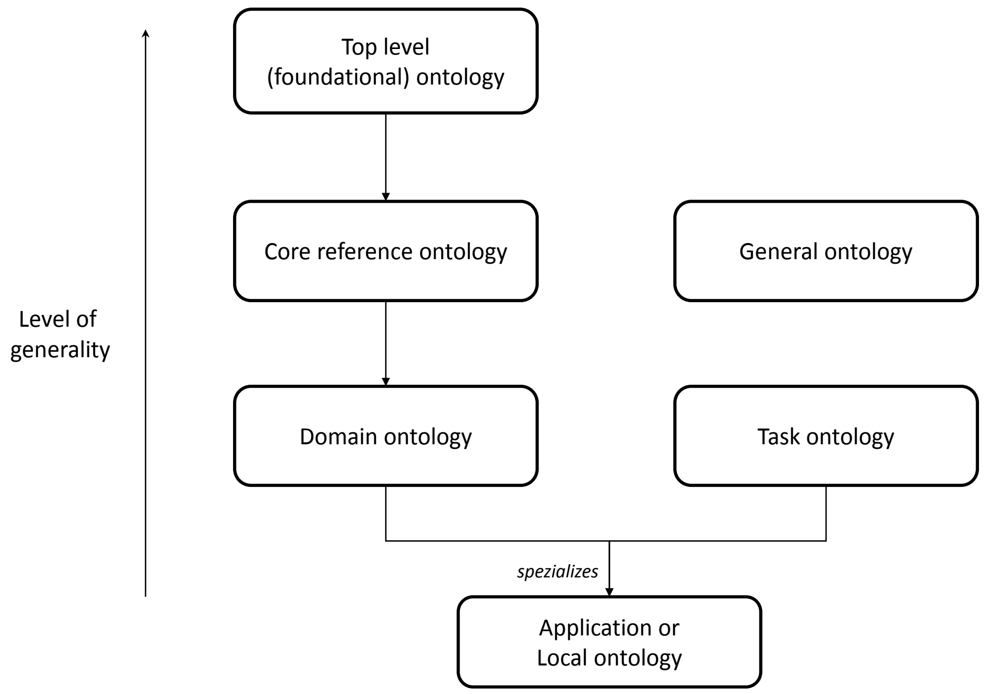


Figure 3.3: Ontology classification based on level of generality (Roussey *et al.* 11)

Top level ontologies

Top level ontologies are used to define others ontologies. They thus apply to various domains. They can be compared with a meta model (Fonseca *et al.* 03). If more specific ontologies, such as for domains or core references, rely on the same foundational ontology, they could more easily be integrated (Roussey *et al.* 11). An example is DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) (Borgo and Masolo 10). Following (Borgo and Masolo 10, Roussey *et al.* 11), DOLCE describes *particulars*. Particulars are entities which have no instance (as opposed to universals, e.g., properties, relations). Particulars can be physical objects (endurants), events (perdurants), qualities, and quales (quality value). Endurants are entities enduring in time, e.g., physical or non-physical objects (social or cognitive entities). Perdurants, e.g., actions, are entities that happen in time and in which endurants participate. Endurants and perdurants have perceivable and measurable inherent properties (qualities). These qualities take a value (quale) within regions of values which are abstract.

Core reference ontologies

Core reference ontologies can be viewed as mid-level ontologies, situated between top-level and domain ontologies (Obrst 10). They reuse concepts specified by top level ontologies, but add concepts and relations which are central to several (neighboring) domains (e.g., car, traffic, aviation) (Prestes *et al.* 13). Thereafter, they can be referred to by those related domains.

General ontologies

General ontologies are not dedicated to a specific domain. Their concepts can be as general as those of core reference ontologies. They might contain hundreds of thousands of concepts and with millions of statements relating the concepts, thereby forming a general ontology whose domain is consensus reality (Roussey *et al.* 11).

Domain ontologies

A domain ontology represents domain-specific knowledge (medicine, automobiles, etc.). Containing domain-specific concepts and relationships, it is thus a specific thesaurus including inference rules, which only apply in this domain (Kaiya and Saeki 06). It specializes the terms introduced in the top-level ontology (Fonseca *et al.* 00).

Task ontologies

Task ontologies describe the conceptualization related to a generic task (Fonseca *et al.* 00). They need to capture task decomposition, which includes control flow, and roles played by entities from the domain in the fulfillment of the task (Martins and de Almeida Falbo 08). Specifically, in (Mizoguchi *et al.* 95), four kinds of concepts are suggested to describe a task: (1) generic nouns representing objects reflecting their roles in the problem solving process, (2) generic verbs representing activities in the problem solving process, (3) generic adjectives modifying the objects, and (4) other concepts specific to the task.

Application ontologies

Application ontologies represent a particular view on a domain. Therefore, they are difficult to be shared due to lacking consensus (Roussey *et al.* 11). In (Fonseca *et al.* 00), this type of ontology is a combination of a specific domain and a task ontology to fulfill a specific purpose.

The task ontology provides knowledge to achieve tasks, the domain ontology describes the knowledge where tasks are applied.

3.3 Semantic integration techniques

Managing the evolution of IS, composed of loosely coupled services presents a great challenge (Lewis *et al.* 08). It is amplified by emerging technologies, such as social networks, smart devices and cars, or else the Internet of Things (IoT) (El-Sheikh *et al.* 13). As introduced in Section 1.1, IS thus operate in complex, dynamic environments. It entails frequent new functional and interface requirements, or partner services, possibly withdrawn or modified without the possibility to intervene.

In such an environment, with continuous changes in complex service compositions, knowledge about individual services is crucial (Gonen *et al.* 15). Incomplete understanding might lead to faulty integrations with uncontrollable ripple effects. As concluded in Section 2.5, it is an unscalable (cognitive) challenge for human experts

Indeed, with hundreds of deployed WS, the task of manually searching and identifying services that satisfy ever changing requirements for integrability can become extremely intensive in manual labor, time spent, and risk of errors (Kourtesis and Paraskakis 10). It ideally warrants more system intelligence in the sense of identification and analysis of input followed by synthesis of possible actions (Zdravković *et al.* 14).

Similar to Chapter 2, the following sections introduce the main semantic integration techniques towards this ideal, namely SAWSDL, WSMO, and OWL-S. Again, we discuss them in light of our running example, reiterated next.

3.3.1 Running example revisited

In Section 2.4.4, we have asked how the system "knows" that a service is preferred over another one. The question could not be answered by selection and integration techniques discussed in Chapter 2. The following sections introduce semantic techniques, supposedly in a better position to do so. We thus adapt the running example presented in Section 2.4.1 to the semantic approach. We use the simple ontology presented in Figure 3.1. We assume that it represents a credit checking (scoring) service from a third-party provider (bold-framed elements) a bank currently uses in its credit loan process. In Figure 3.4, we show a competitive third-party credit checking service. It has the same setup as the current service, except it uses a new scoring algorithm with more reliable results (grey pane). Clearly, henceforward the bank would want to use the new service.

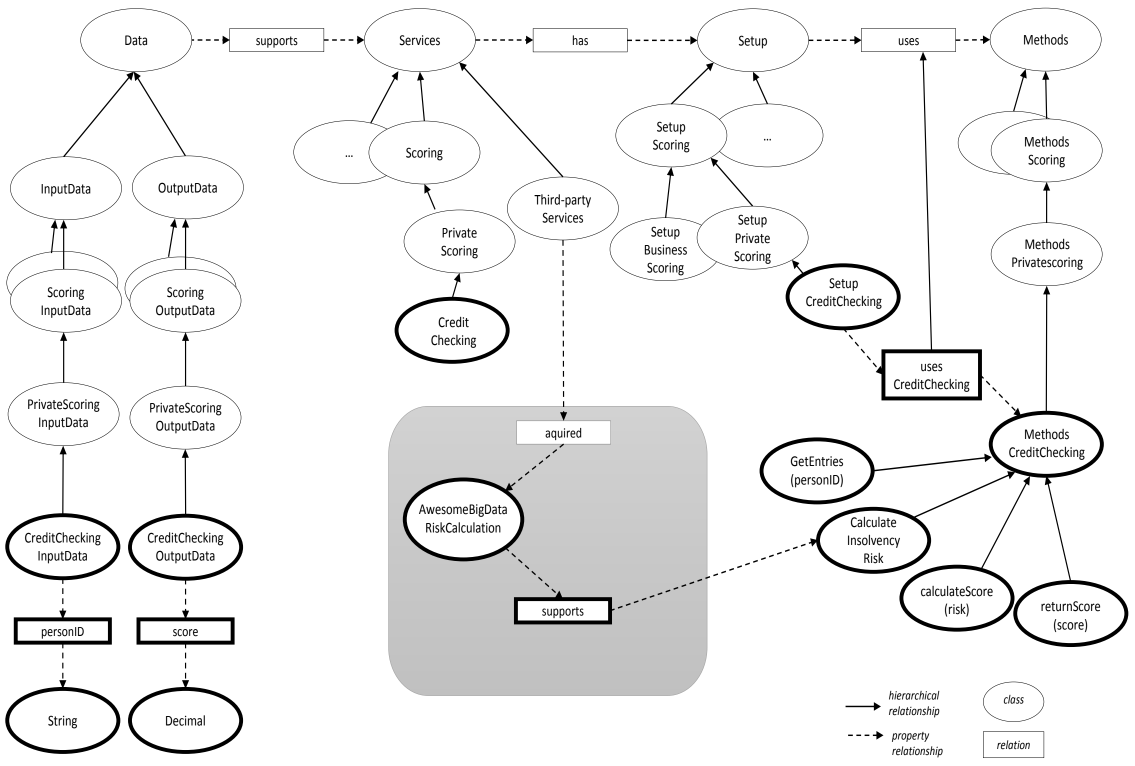


Figure 3.4: Ontology for a competitive credit checking service

We briefly discuss this challenge for each semantic integration approach presented below. We start with SAWSDL.

3.3.2 SAWSDL

Semantic Annotations for WSDL (SAWSDL) is an extension to WSDL and provides a semantic annotation mechanism. Whereas WSDL specifies how a message looks like, the SAWSDL specifies what a message means. Specifically, it uses pointers to semantic concepts from within the syntactic WSDL-based service description (Fensel *et al.* 11). Its major constructs are *modelReference* and *schemaMapping*. The latter is split into *liftingSchemaMapping* and *loweringSchemaMapping* (Fensel *et al.* 11, Wei *et al.* 11).

modelReference is used to point to semantic concepts. It can be used for <interfaces>, <operations>, <faults>, as well as XML Schema <elements>, <complex types>, <simple types>

and <attributes> (cf., example listing below). modelReference is typically used in automated service discovery and composition. It is recommended by the SA-WSDL working group that the reference contains an URI identifying the concepts in an external knowledge-representation or term definition system (Lemcke 10). Cf., following example listing (Lausen and Farrell 07):

```
<wsdl:interface name="Order"
  sawsdl:modelReference=
   "http://example.org/categorization/products/electronics">}
 <wsdl:operation name="order"
    pattern="http://www.w3.org/ns/wsdl/in-out"
   sawsdl:modelReference=
    "http://www.w3.org/2002/ws/sawsdl/spec/ontology/
    purchaseorder#RequestPurchaseOrder">
  <wsdl:input element="OrderRequest" />
  <wsdl:output element="OrderResponse" />
 </wsdl:operation>
</wsdl:interface>
```

schemaMapping is used to specify data transformations between the XML data structure of messages and an associated semantic model (Fensel *et al.* 11). liftingSchemaMapping transforms XML data from a Web service message into a semantic model. loweringSchemaMapping transform data from a semantic model into an XML message (cf., Fig. 3.5).

schemaMapping helps when data structure mediation is needed to support invocation of a Web service. For example, a client may have first name and last name among its data. These values need to be concatenated in the message to the Web service. A lowering schema mapping would turn the client's semantic data into XML while performing concatenation to produce the full name (Lausen and Farrell 07).

SAWSDL by itself does not specify any actual types of semantics. It is agnostic to the knowledge representation formalism one adopts for representing service characteristics.

Running example In the scenario described, we would have two competing services c_{old} and c_{new}. A client request (step 3, Capture personal information) would need to automatically discover and use c_{new} instead of c_{old}. To us, there are a number of considerations:

- In a WS-based environment, it cannot be assumed that the new service's <interface>, and consequently <operations>, <messages>, <types> are different from the old service.

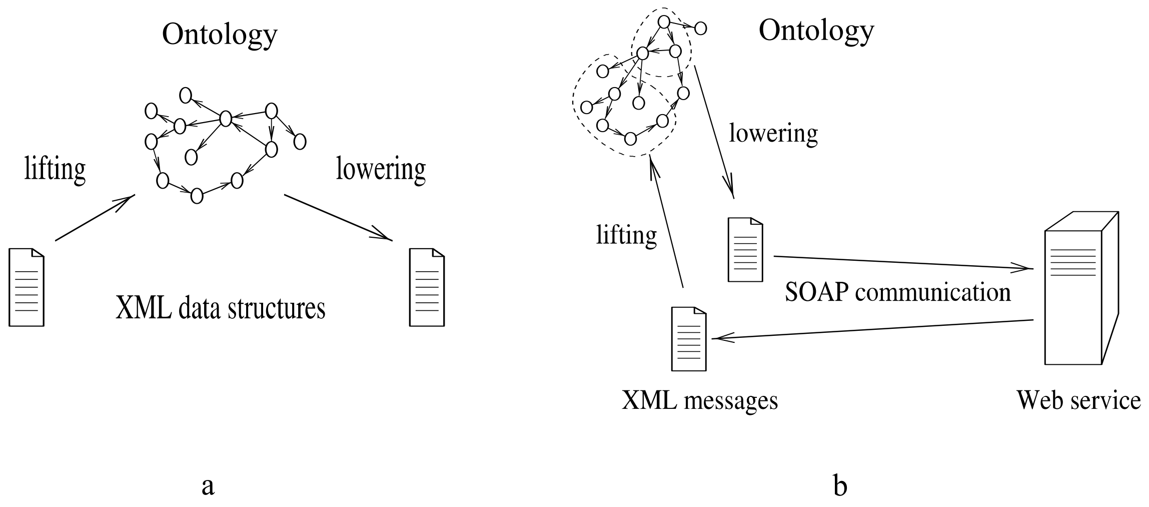


Figure 3.5: SAWSDL-based data lifting and lowering. a) for WS communication. b) for data mediation

Although they are, e.g., due to a different backend processing, to a service requester they might appear alike.

- The implied semantics of a service might be too specific or dynamically changing as to be describable by domain or foundational ontologies as external semantic representation. In turn, application ontologies might be too specific to support the service discovery. It does not necessarily represent a shared view. For example, in the targeted services, there might simply be no semantic correspondence, e.g., through concept subsumption, to the requested terms.

- In any case, human experts would need to analyse the external ontology landscape to identify appropriate ones. Iteratively, they need to recheck concepts, and possibly point to new URIs.

In light of these points, a SAWSDL approach seems more appropriate within specific and rather stable (closed) knowledge domains. For requester services, it implies the need to already tune to the semantic descriptions, thus, the external ontological representations, probably used by target services. To us, this makes the technique unsuitable for (open) environments with fast changing semantics. It does not scale.

3.3.3 OWL-S

OWL-S is a top level ontology for the description of semantic WS expressed in OWL. It defines three central elements, namely *ServiceProfile*, *ServiceGrounding*, *ServiceModel* (cf., Fig. 3.6). A service is declared by creating an instance of the Service concept (Burstein *et al.* 04).

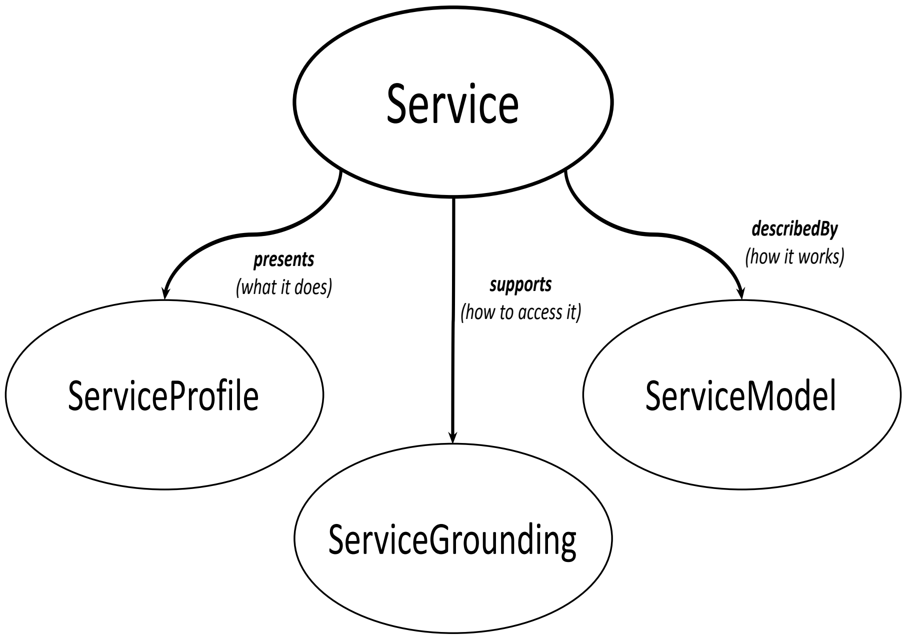


Figure 3.6: OWL-S Service Ontology (Burstein *et al.* 04)

ServiceProfile

The ServiceProfile describes what the service does by means of inputs, outputs, preconditions, and effects (IOPEs). It is further detailed in Figure 3.7.

The service profile thus represents capabilities of a service (Fensel *et al.* 11). Specifically, *serviceName*, *textDescription*, and *serviceProduct* provides a simple service description and contact information of the service provider. *Inputs*, generated *outputs*, external *conditions* constraining the execution of the service, and effects (*results*) that change these conditions subsume the functional aspects of the service profile. *ServiceCategory* categorizes the service. *ServiceParameter* comprises a list of service parameters containing any kind of information (e.g., maximum response time, geographical availability, quality rating). Cf., following example

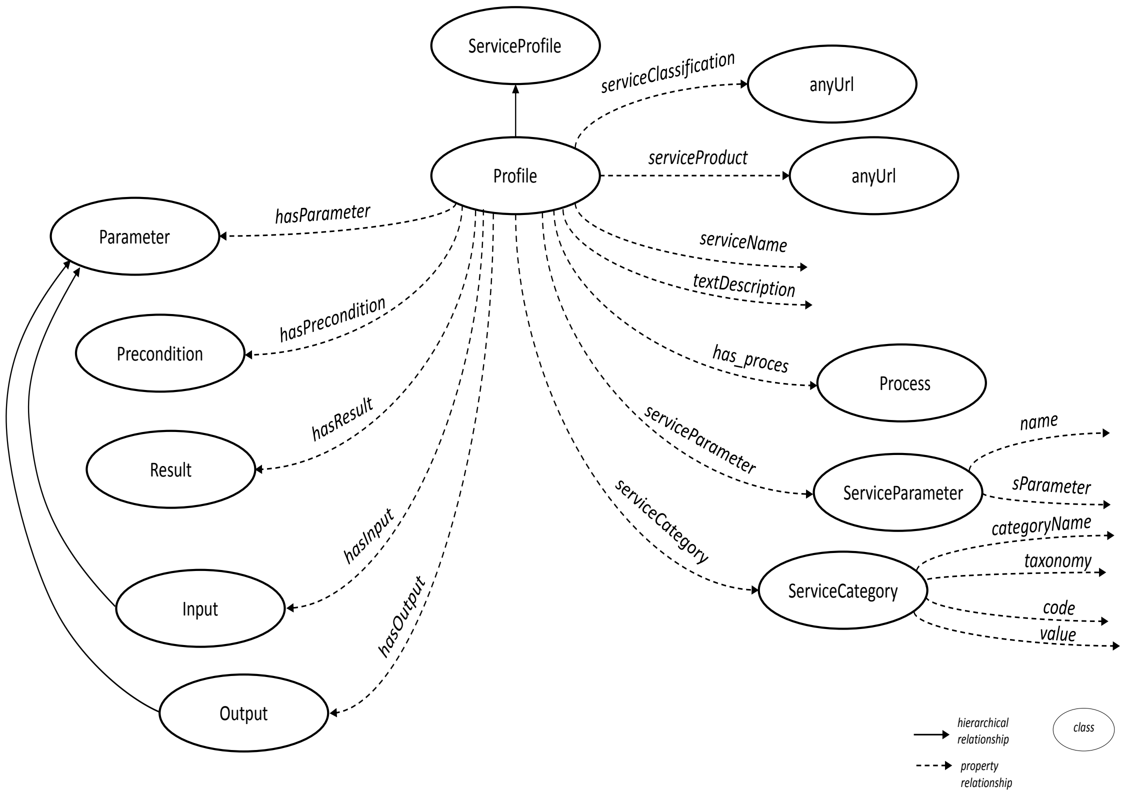


Figure 3.7: OWL-S Service Profile (Burstein *et al.* 04)

listing[1]:

```
<profile:serviceName>CreditChecking</profile:serviceName>
<profile:textDescription> ... </profile:textDescription>
<profile:contactInformation>
 <actor:Actor rdf:ID="ContactXY">
  <actor:name> ... </actor:name>
  <actor:title> ... </actor:title>
  <actor:phone> ... </actor:phone>
  <actor:fax> ... </actor:fax>
  <actor:email> ... </actor:email>
  <actor:physicalAddress> ... </actor:physicalAddress>
  <actor:webURL> ... </actor:webURL>
 </actor:Actor>
```

[1]adapted from http://www.daml.org/services/owl-s/1.1/examples.html

```
</profile:contactInformation>
<profileHierarchy:deliveryRegion rdf:resource=
   "http://www.daml.org/services/owl-s/1.2/Country.owl#Switzerland"/>
<profile:serviceCategory>
 <addParam:NAICS rdf:ID="NAICS-category">
  <profile:value>
   Consumer Lending
  </profile:value>
  <profile:code>
   522291
  </profile:code>
 </addParam:NAICS>
</profile:serviceCategory>

...

<profile:hasInput rdf:resource= "http://www.creditregistry.com/
       CreditChecking.owl#creditScoreRequest_In"/>
<profile:hasOutput rdf:resource="http://www.creditregistry.com/
       CreditChecking.owl#getCreditScore_Out"/>
<profile:hasPrecondition rdf:resource="http://creditregistry.org/
       CreditChecking.owl#personIDExists"/>
<profile:hasResult rdf:resource="http://creditregistry.org/
       CreditChecking.owl#hasScoringResult"/>
```

In the listing, input, output, precondition, and effect refer to concepts encoded in some ontology. Note, that these can be private, public, top level, domain-specific ontology, etc. It is indicated by the prefix “#”. If a service requester seeks a certain service, it may now look for semantic similarity of those concepts with the requested element. Specifically, this could happen as follows (adapted from (Farrag *et al.* 13)): a requested concept is called C_R and may have three properties P_1, P_2, and P_3 (e.g., label, relationship to another concept, data type). It is compared against an advertised concept C in the service ontology. Relative to C_R, C can have four different positions in a concept hierarchy:

a) Identical relation: C and C_R have the same properties

b) Super relation: C is a parent of C_R.

c) Sub relation: C is a child of C_R.

d) Neighbor relation: C and C_R have some common properties.

In Figure 3.8, these possibilities are shown. Depending on the requester's preference, a selection mechanism may thus screen several competing services and assign a ranking value to each one, such that "identical" has the highest ranking, followed by"Super," "Sub," and "Neighbor."

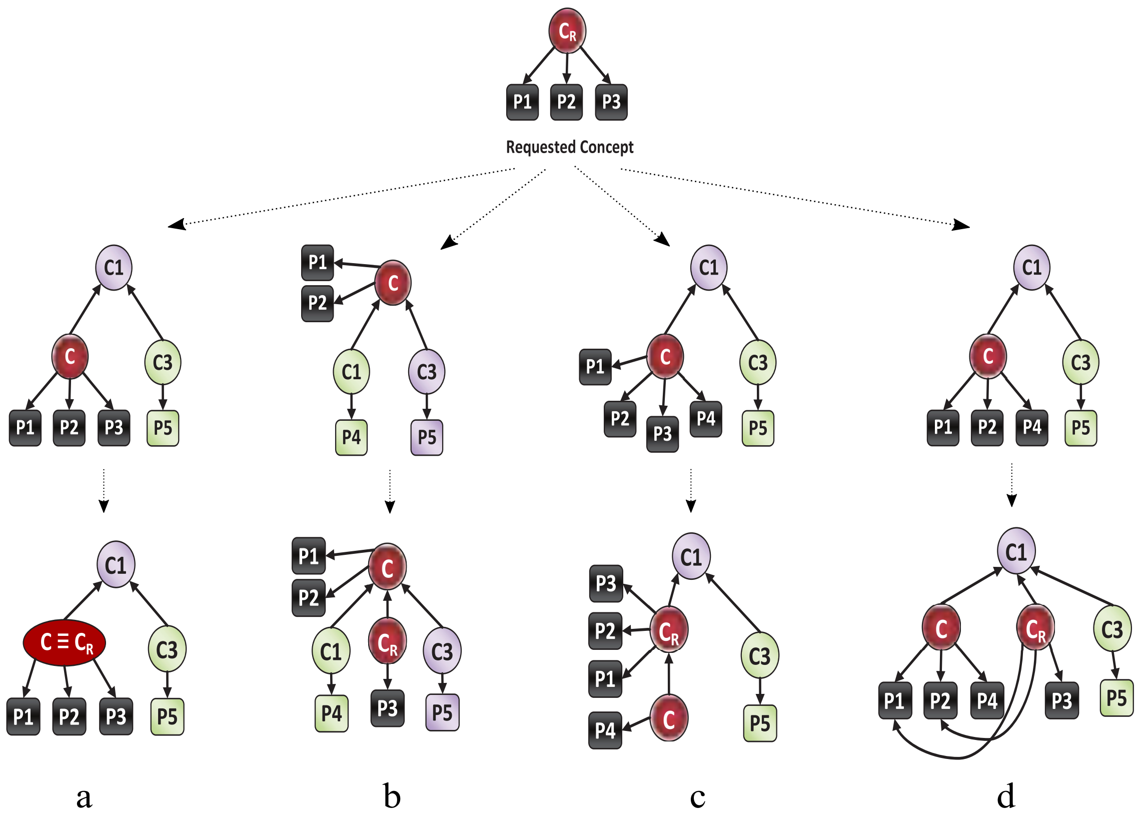


Figure 3.8: Concept matching (adapted from (Farrag *et al.* 13))

ServiceModel

The ServiceModel describes all the processes the service is composed of, how these processes are executed, under which conditions they are executed. It answers the question, how the service works (Farrag *et al.* 13) (cf., Fig. 3.9).

Atomic processes represent directly invocable operations. They are executed in a single step and cannot be further refined (Fensel *et al.* 11). An atomic process takes an input message,

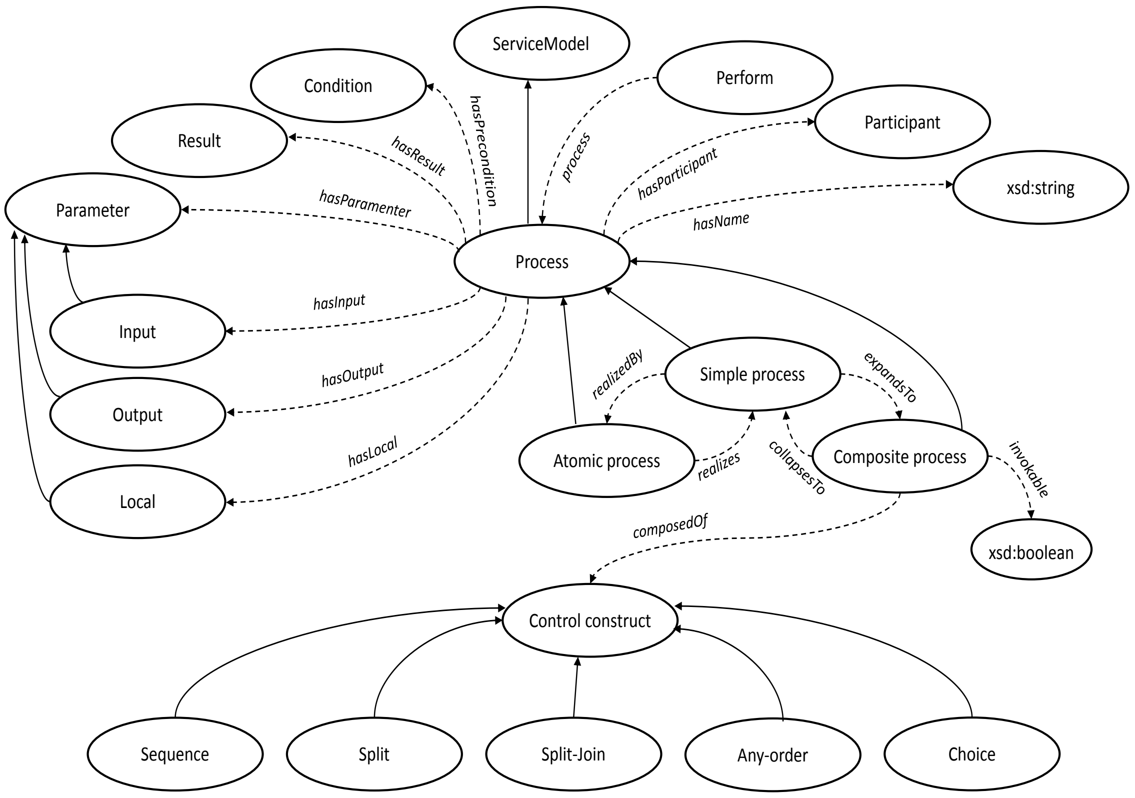


Figure 3.9: OWL-S Service Model (Burstein *et al.* 04)

and returns an output message. They have an associated grounding to a service specification (e.g., WSDL), so service requesters can construct input and output messages. *Simple processes* also perform single-step executions. However, they do not have associated grounding and are not invocable (Burstein *et al.* 04). The simple processes are used as wrappers, offering specific ways to use an atomic process, or to simplify the representation of a composite process for planning and reasoning purposes (Fensel *et al.* 11). *Composite processes* represent more complex decomposable processes (composite or non-composite). The decomposition is defined with control constructs (e.g., sequence). A composite process describes a behavior the client can enact by sending and receiving a set of messages. Cf., following example listing[1]:

```
<process:ProcessModel rdf:ID="CreditChecking_ProcessModel">
 <process:hasProcess rdf:resource="#CreditChecking_Process"/>
 <service:describes rdf:resource="http://www.daml.org/services
  /owl-s/1.0/CreditCheckingService.owl#CreditChecking"/>
```

[1]adapted from http://www.daml.org/services/owl-s/1.1/examples.html

```
</process:ProcessModel>
<process:CompositeProcess rdf:ID="CreditCheckingProcess">
 <process:composedOf>
  <process:Sequence>
   <process:components rdf:parseType="Collection">
    <process:AtomicProcess rdf:about=
       "#LoginCreditRegister"/>
    <process:AtomicProces rdf:about=
       "#getScoreFromCreditRegister"/>
   </process:components>
  </process:Sequence>
 </process:composedOf>
</process:CompositeProcess>

<process:AtomicProcess rdf:ID="LoginCreditRegister">
 <process:hasInput rdf:resource="#AcctName_In"/>
 <process:hasInput rdf:resource="#Password_In"/>
</process:AtomicProcess>

...

<process:Input rdf:ID="AcctName_In">
 <process:parameterType rdf:resource="http://www.daml.org/
       services/owl-s/1.0/Concepts.owl#AcctName"/>
</process:Input>
<process:Input rdf:ID="Password_In">
 <process:parameterType rdf:resource="http://www.daml.org/
       services/owl-s/1.0/Concepts.owl#Password"/>
</process:Input>

...
```

Again, defined concepts may stem from some ontology. Different from the service profile, the focus, here, is semantic execution support.

ServiceGrounding

The ServiceGrounding describes how the service can be accessed. It specifies how the abstract OWL-S service representations, i.e., service profile and service model, are mapped to the concrete service description elements, e.g., WSDL, required for interacting with the service (inputs and outputs of an atomic processes) (Fensel *et al.* 11).

For the WSDL specification, the following grounding rules exist: OWL-S atomic process → WSDL operation, OWL-S input/output → WSDL message, OWL-S input/output types → WSDL extensible notion of abstract type, which can be used in WSDL message specifications (Fensel *et al.* 11).

Cf., example of the WSDL/OWL-S grounding in the following example listing[1]. It shows grounding into the WSDL-based service described in Section 2.4.2:

```
<grounding:WsdlGrounding rdf:ID ="Grounding_CreditChecking">
 <service:supportedBy rdf:resource="#CreditChecking"/>

 <!--Collecton of all the groundings specifications-->
 <grounding:hasAtomicProcessGrounding
       rdf:resource=
           "#WsdlGrounding_LoginCreditRegister"/>
 <grounding:hasAtomicProcessGrounding
       rdf:resource=
           "#WsdlGrounding_getScoreFromCreditRegister"/>
</grounding:WsdlGrounding>

<!--Grounding for atomic process getScoreFromCreditRegister-->
<grounding:WsdlAtomicProcessGrounding
       rdf:ID="WsdlGrounding_getScoreFromCreditRegister">
 <grounding:owlsProcess
       rdf:resource="#getScoreFromCreditRegister"/>

 <!--Reference to the corresponding WSDL operation-->
 <grounding:wsdlOperation
       rdf:resource="#getScoreFromCreditRegister"/>
```

[1]adapted from http://www.daml.org/services/owl-s/1.1/examples.html

```
  <!--Reference to the WSDL input message-->
  <grounding:wsdlInputMessage>
   <xsd:anyURI rdf:value=
   "&CreditCheckingGroundingWSDL;
      #getScoreFromCreditRegisterSoapIn"/>
  </grounding:wsdlInputMessage>


  <!--Mapping of OWL-S inputs to WSDL message parts-->
  <grounding:wsdlInputs rdf:parseType="Collection">
   <grounding:WsdlInputMessageMap>
    <grounding:owlsParameter rdf:resource="#personID"/>
    <grounding:wsdlMessagePart>
     <xsd:anyURI rdf:value="&CreditCheckingWSDL;#personID"/>
    </grounding:wsdlMessagePart>
   </grounding:WsdlInputMessageMap>
  </grounding:wsdlInputs>
  <grounding:wsdlReference>
   <xsd:anyURI
      rdf:value="http://www.w3.org/TR/2001/NOTE-wsdl-20010315"/>
  </grounding:wsdlReference>
 </grounding:WsdlAtomicProcessGrounding>

 <grounding:WsdlOperationRef rdf:ID="getScoreFromCreditRegister">
  <!--locate interface to be used -->
  <grounding:interface>
   <xsd:anyURI
     rdf:value="&CreditCheckingGroundingWSDL;#getCreditScore"/>
  </grounding:interface>
  <!--locate operation to be used -->
  <grounding:operation>
   <xsd:anyURI rdf:value=
     "&CreditCheckingGroundingWSDL;#getScoreFromCreditRegister"/>
  </grounding:operation>
 </grounding:WsdlOperationRef>
```

. . .

By grounding OWL-S in WSDL, advantages from both languages are combined. For example, WSDL is unable to express the semantics of an OWL class or its relationships. In turn, OWL-S is unable to express the binding information that WSDL captures.

Running example The OWL-S ontology is a top level OWL ontology for semantically describing fundamental service aspects (Izza *et al.* 08). It thus provides a general description framework. Typically, services, especially inputs, outputs, preconditions, and effects (IOPEs), are described by referring to adequate external knowledge representations, e.g., application, domain, or core ontologies (Klusch *et al.* 09, Shin *et al.* 09, Zuñiga *et al.* 14). With regards to the running example, we make similar remarks as for SAWSDL:

- IOPEs are the main vehicle to represent what a service does. If backend processing worth annotating is not captured in those through adequate external semantic references, it cannot adequately be advertised and thus selected. In other words, the OWL-S Service concept (cf., Fig. 3.6) may have the same specification for two services, despite them being different. It relates to the next point.

- Referenced ontologies might be too specific (e.g., organization-internal ontologies) to support the service discovery and selection, as they do not necessarily represent a shared view, i.e., a requesters view. If, on the other side, they are too general, the service might not be discovered for a competitive (hence, new) trait.

The dilemma of standardization vs. flexibility is, again, apparent. To us, OWL-S, though more sophisticated than SAWSDL, still implies coherent collaboration networks. They may serve as stabilizing element to enforce minimal formal semantic standards by means of domain or core reference ontologies. Indeed, with respect to semantic annotation techniques such as OWL-S, (Van Der Aalst *et al.* 03) states that still a well-founded least common denominator is needed to counteract ambiguity and non-determinism. This denominator can then be used to apply the strength of the annotation technique. For our example, pertaining to dynamic, unforeseeable, and competitive markets, we perceive it as problematic.

3.3.4 WSMO

The WS Modeling Ontology (WSMO) is a meta model (cf., Fig. 3.10). It provides a general framework for WS model descriptions which result in enactable WS (Lemcke 10). Its goal is to enable the total or partial automation of tasks occurring when using WS. It includes (De Bruijn *et al.* 09):

- discovering services to fulfill some task
- selecting services if more than one could accomplish the task
- composing services if the tasks are more complex
- resolving service heterogeneity on data and process level
- invoking services

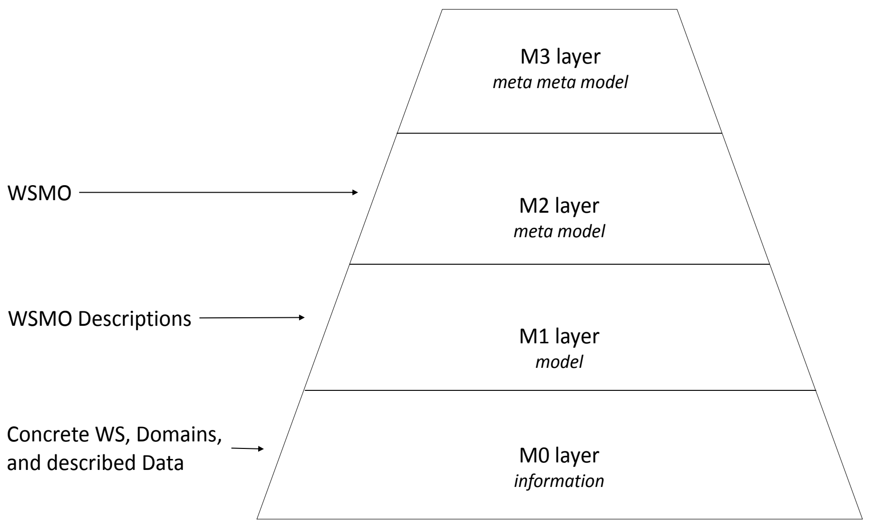


Figure 3.10: WSMO positioning (De Bruijn *et al.* 09)

The approach is based on (1) strongly de-coupling the various components that realize a distributed application, and (2) mediation enabling WS to communicate in a scalable manner (Sheng *et al.* 14). Different to OWL-S, which takes a service point of view to describe service activities, WSMO centers around a client's view and its goals. The client may be either human or a requester service (Kamaruddin *et al.* 12).

WSMO (De Bruijn *et al.* 09, Fensel *et al.* 11, Wang *et al.* 12) relies on four top level elements: *ontologies* to provide the terminology, *goals* to define problems to be solved by the WS, *Web services descriptions* defining aspects of a WS, and *mediators* to handle interoperability issues (cf., Fig. 3.11). They are detailed below.

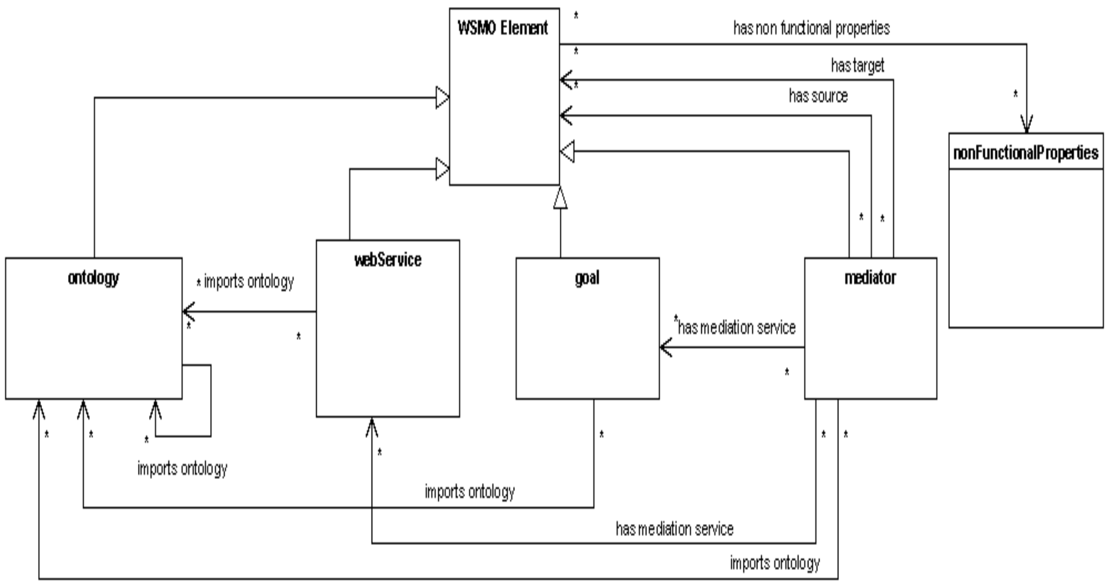


Figure 3.11: Upper WSMO Elements (De Bruijn *et al.* 09)

Ontologies

Ontologies are described at a meta-level. They provide formal and explicit specifications of the vocabulary used by other modeling elements in WSMO. All top levels elements within the WSMO meta-model can use the *importsOntology* statement to import ontologies that contain the relevant concepts needed to build a description.

Goals

A goal is defined as a client's objective or requirement when consulting a WS. For example, it contains the requested capability including input and output requirements. Optionally, it might also hold interface specification, i.e., the interaction pattern the service requester expects. Furthermore, desired non-functional properties could be described.

Web Service descriptions

Web service descriptions semantically annotate the WS themselves. They can be seen as the counterpart to Goals, against which they are compared. It includes three parts, namely *capability*, *interface*, and *non-functional information*.

Capabilities, i.e., functional aspects, of the offered service are modelled in terms of preconditions, assumptions, postconditions and effects. Preconditions describe conditions prior to executing a service. They define requirements on the input (e.g., type). Assumptions are conditions for proper execution (e.g., credit card balance sufficient to purchase an item). Postconditions describe the information space after execution. They define requirements on the output. Effects are guaranteed conditions in the real world after execution (e.g., a book is delivered). Interfaces are data-centric, similar to WSDL. They thus provide details about accessing the service's operations. Non-functional information comprise meta-data, such as performance parameters (reliability, security, availability, ping time, etc.).

Mediators

Mediators resolve heterogeneity problems. They define mappings, transformations, or reductions between elements involved in the semantic descriptions. Four mediators exists, namely ggMediators, ooMediators, wgMediators, and wwMediators. *ggMediators* link a source goal and a target goal. They can use ooMediators to solve differences in the terminology used to define these goals. Goals can also be linked to ggMediators. It enables reuse of multiple goals to define a new one. *ooMediators* import ontologies and resolve possible representation mismatches between them, such as differences in their conceptualizations. *wgMediators* link a WS to a goal. This link represents the fulfilment (partial or complete) of the goal by the WS. wgMediators can use ooMediators to resolve heterogeneity problems between the WS and the goal. *wwMediators* link two WS. Again, ooMediators might be used to overcome heterogeneity problems between the WS.

WSMO's principle mode of operation

First, services are discovered by matching capabilities and goals. Obviously, this is based on semantic descriptions. Second, if one or more services are identified, interface specification and requirements must match. If the request specifies a login operation *before* a buying operation, the interface must comply with this order, i.e., interaction pattern. Third, the interface's possible orchestration description is matched with the requester's specifications. From 2.4.3, we know that an orchestration specifies which services a service relies upon to provide functionality. A requester may prescribe a book delivery of three days or less. The service would need to comply with this. Fourth, if several services provide the above, the non-functional parameters are compared and matched. Fifth, *ceteris paribus*, the service with the lowest price is selected and automatically invoked based on the service interface description (similar to WSDL).

For each phase, semantic matching can be used. The matching is similar to OWL-S (cf., Fig. 3.8). Some examples are given below (adapted from (De Bruijn *et al.* 09)):

Capability/Goal matching: Given an ontology $\mathcal{O}$, a WS (concept) $\mathcal{WS}$, and a goal (concept) $\mathcal{G}$, the following matching possibilities are distinguished:

a) *Exact match* - the requested elements are fulfilled by all and only those WS elements ($\mathcal{G} \equiv_{\mathcal{O}} \mathcal{WS}$).

b) *Subsume match* - the WS provides some but not all of the requested elements ($\mathcal{WS} \subseteq_{\mathcal{O}} \mathcal{G}$).

c) *PlugIn match* - all requested elements are provided by the WS, but the WS provides even more ($\mathcal{G} \subseteq_{\mathcal{O}} \mathcal{WS}$).

d) *Intersection match* - some requested elements are provided by WS ($\mathcal{WS} \cap \mathcal{G} \not\subseteq_{\mathcal{O}} \perp$).

e) *Intersection non-match* - no requested elements are provided by WS ($\mathcal{WS} \cap \mathcal{G} \subseteq_{\mathcal{O}} \perp$).

Input/Output (Preconditions/Postconditions) matching: WS (goal) inputs and outputs are similar to method or function signatures. They denote message formats sent to, and returned by, the service. Other than for signatures, here the message formats are described semantically, denoting intention. Note that the concepts describing inputs (outputs) are part of the preconditions (postconditions).

From individual WS inputs and outputs defined as $\mathcal{WS}_{I_i}$ and $\mathcal{WS}_{O_j}$, with $i \leq m$ and $j \leq n$ (similar for $\mathcal{G}$) follows the overall set of inputs and outputs $(\mathcal{WS}_I, \mathcal{WS}_O, \mathcal{G}_I, \mathcal{G}_I)$ with $(\mathcal{WS}_I \equiv \exists hasInput.\mathcal{WS}_{I_1} \sqcap \ldots \sqcap \exists hasInput.\mathcal{WS}_{I_m})$, $(\mathcal{WS}_O \equiv \exists hasInput.\mathcal{WS}_{O_1} \sqcap \ldots \sqcap \exists hasInput.\mathcal{WS}_{O_n})$, $(\mathcal{G}_I \equiv \exists hasInput.\mathcal{G}_{I_1} \sqcap \ldots \sqcap \exists hasInput.\mathcal{G}_{I_k})$, and $(\mathcal{G}_O \equiv \exists hasInput.\mathcal{G}_{O_1} \sqcap \ldots \sqcap \exists hasInput.\mathcal{G}_{O_l})$. The following matchings possibilities are distinguished:

a) *Signature full match* - all inputs needed by the WS are provided by the goal, and all outputs requested by the goal are provided by the WS ($\mathcal{G}_I \equiv_{\mathcal{O}} \mathcal{S}_I$, and $\mathcal{WS}_O \equiv_{\mathcal{O}} \mathcal{WS}_O$).

b) *Signature output match* - all outputs requested by the goal are provided by the service ($\mathcal{WS}_O \subseteq_{\mathcal{O}} \mathcal{WS}_O$).

c) *Partial signature match* - some outputs requested by the goal are provided by the service $\mathcal{WS}_O \cap \mathcal{G}_O \not\subseteq_{\mathcal{O}} \perp$.

d) *Signature non-match* - none of the outputs requested by the goal are provided by the service $\mathcal{WS}_O \cap \mathcal{G}_O \subseteq_{\mathcal{O}} \perp$.

Note, that the focus here is on output descriptions. The authors ((De Bruijn *et al.* 09, Fensel *et al.* 11)) do not expect requesters to thoroughly know, let alone specify, desired inputs. Requesters are more interested in the desired outputs. However, the above formalism is similar for input matchings.

Running example WSMO aims at a unifying framework, a meta model, defining the semantics and the syntax with which to describe the semantics of web services (Izza 09). It applies to all relevant WS aspects, such as goals, capabilities, inputs, and outputs. It provides mediators for bridging ontological gaps among different meaning systems.

Concerning the running example, i.e., dynamically exchanging the credit checking service, we think that WSMO provides the most compelling option so far. The definition of goals and capabilities suits our context. It emphasizes the requester's point of view being matched against the service's capabilities. To us, this concept is more powerful than merely relying on formalized IOPEs, such as in OWL-S. The competitive credit checking service's capability part can be extended with a reference to the sophisticated algorithm. The requester's goals though must also be extended with a similar reference. Here, the questions arises how a requester might know about this new algorithm and how to circumscribe it.

However, WSMO is a framework that already unifies at the meta-model layer. It is difficult to use without broad adoption. It concerns service providers and requesters alike as goals and capabilities have to be under the same umbrella technology, which is WSMO. In a heterogeneous environment, one meta model "to rule them all" though is less likely adopted. Indeed, (Kamaruddin *et al.* 12) states that the WSMO (as well as OWL-S) is not universally applicable, but depends on the purposes. The authors' survey on WSMO and OWL-S usage suggests specific, relatively stable domains, e.g., governmental. In other words, if the requester does not understand the WSMO meta model, the more sophisticated service might not be recognized.

Another issue, which clearly applies to all techniques introduced so far, is the symbolic approach itself. As stated within Section 3.1, eventually they solely rely on arbitrary symbolic shapes. Consequently, it also applies the matching act. If symbols are not equal, there is no match. Compensating mediators are based on the same approach. They therefore do not address this point. It will be further detailed in the remainder of this document (e.g., Sect. 5).

3.4 Discussing the gap

Current industrial integration techniques (cf., Chap. 2) are entirely based on syntax. The meaning of interfaces is defined by human experts who implement them. Granted, some degree of adaptability is offered by means of rule engines. However, it only applies to a predefined set of anticipated composition paths (cf., Fig. 3.12).

It is useless for unforeseeable changes. Those need expert intervention. Eventually, these experts constantly negotiate and agree on input and output terms, e.g., as different service providers might have different assumptions about them; monitor and rectify changing interface specifications, e.g., regarding data types, field lengths; and need to understand and translate changing business requirements in order to adapt services compositions and data flows.

Due to the syntactic nature, for many authors (Izza 09, Hoang and Le 09, Fensel *et al.* 11, Hoang *et al.* 14), the above mentioned integration techniques are blind to the semantics of services. They merely regulate information and meta-data. The authors advocate to use a formal-semantic approach as described in this chapter. For them, semantic descriptions of services are necessary to establish interoperability without human intervention, which otherwise is costly, rapidly obsolete, or non-reusable in an environment with the dynamic contexts described so far. In short, it shall enable machines to understand the meaning of services to accomplish the tuple ⟨*discover, select, resolve (heterogeneity), compose, invoke*⟩ (cf., again Fig. 1.2).

Clearly, the usage of meaning systems and automatic reasoning capabilities (cf., Fig. 3.8 and 3.10) improve on those aspects over syntactic approaches. However, at several occasions we mentioned shortcomings, such as:

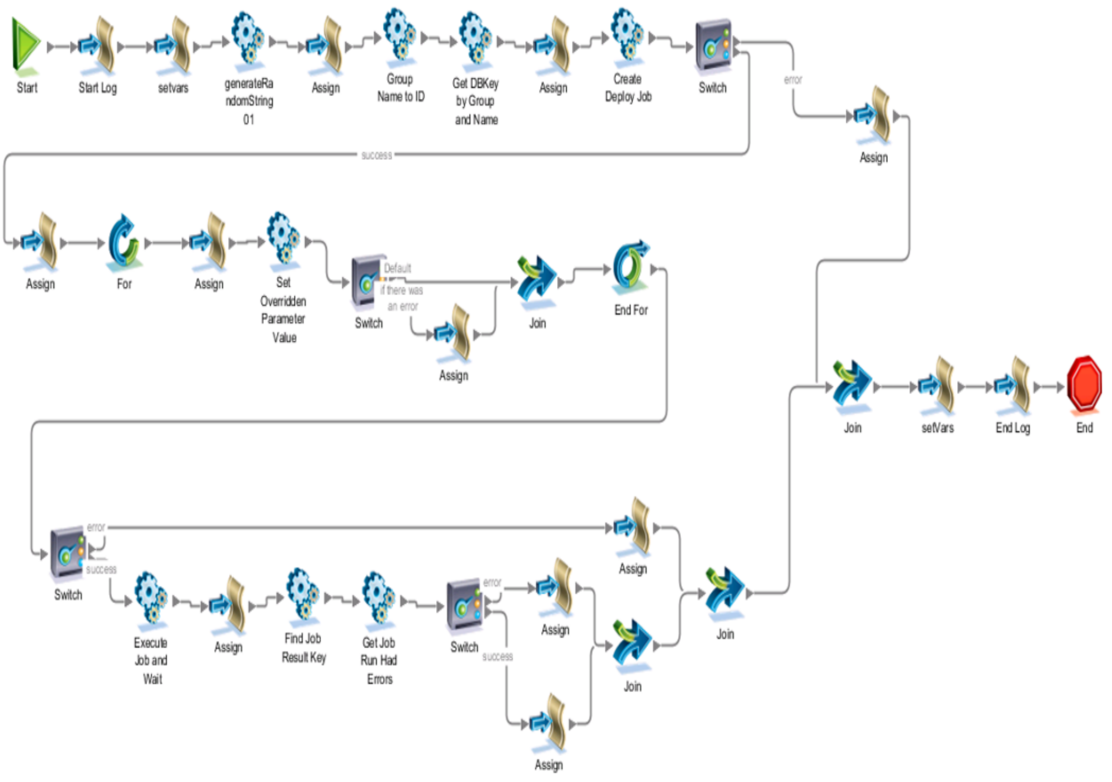


Figure 3.12: Commercial workflow engine with workflow example. Contingent execution paths are predefined.

- Simple structure of semantic services descriptions (e.g., OWL-S's IOPEs). They are possibly insufficient or too restrictive to capture distinctive aspects of enterprise-level services (e.g., Salesforce's CRM).
- Practicality only for simple services, expressible by the standards.
- Single, specific meta model (e.g., WSMO), implying that requesters and providers respect it "to the letter." Only then, they can adequately formulate requirements[1] to find each other.
- Tendency to assume stable collaborative networks for proper operation. However, this defies the purpose of coping with the reality of an unforeseeable, independent service landscape.

[1]For example goals, resp. capabilities.

- Purely symbolic matching, possibly missing out on “close-to” matches if goals or requirements are ill- or non-defined, or adhere to different symbol system (e.g., Arabic).
- Formal-semantic systems, still being symbol systems with no true representation of meaning. Therefore, flexibility is still predefined, although less than the case with syntactic approaches.

Indeed, those points reflect in (Oberle *et al.* 05), conveying a similar scepticism. To them, it is unclear, what kind of powerful machinery could constitute a semantic model allowing for full automation, and indeed, such full automation seems outside the scope of software systems within the foreseeable future. The authors advocate that semantic management of WS should not try to produce full automation of all WS management tasks. That would presuppose an understanding of the world, too difficult to be modelled explicitly. Consequently, the described symbol-semantic, i.e., explicit, techniques are always too abstract or coarse-grained, and insufficient for continuous, adequate and automated service selection and integration.

Eventually, these issues limit the usage of symbol-semantic approaches under heterogeneous, unforeseeable conditions. To us, it follows that the first research questions, namely,

Can ontologies stand-alone, that is, in isolation, be used for an intelligent service selection and integration method?

can not be answered satisfactorily with yes, if we only look at ontologies.

Clearly, semantic descriptions (i.e., ontologies) can be, and are used, as means for service selection and integration. However, an *intelligent* approach presupposes the proactive identification and understanding of *unknown* input and analysis towards synthesis of possible actions to perform in response to the understood input (Zdravković *et al.* 14). As stated in Section 3.4, this is necessary for continued understanding of what a good service is, selection of the best[1] service, and integration of the selected service. Again, the specific fact of restrictive meta models, and the general fact of relying on purely symbolic representations would not allow for such behavior.

[1]For a definition of “good” and ”best”, cf., Chapter 2.

During the next chapters, we therefore combine the above symbol-based ontology approach with a nonlinear data representation approach, the LRAAM (Labelled Recursive Auto-Associative Memory). We advocate that the combination of explicit (symbolic) and reduced (subsymbolic) semantic approaches brings us closer to what we understand by an intelligent SSC. Specifically, with an LRAAM, an entire arbitrary-sized graph structure of a semantic description, i.e., ontology, can potentially be compressed into a fixed-size neural network representation (de Gerlachey *et al.* 94, Sperduti 93). This distributed, *inner* representation is then transformed into a distance measure to better (i.e., holistically) compare semantic specifications, e.g., representing business requirements (i.e., goals) and services (i.e., capabilities).

4

Global service selection and composition algorithm

In Chapter 3, we have made the case that under changing conditions, formal-semantic service descriptions improve automated service integration over purely syntactic approaches. Specifically, machine reasoning based on ontological descriptions allows for better, and automatic, matching of the right concepts. It may for example concern input/output or requirement/functionality relations (cf., (Antoniou and van Harmelen 08), or (Antoniou and van Harmelen 09) for further details).

However, we also discussed issues limiting its usage under heterogeneous, unforeseeable conditions. These limits are due to the assumption of restrictive meta-models to apply throughout, or to the arbitrary nature itself of symbolic signs with no inner meaning.

Within the next sections, we aim at attenuating this conflict. To this end, we propose a global formal-semantics based service selection and composition algorithm. For convenience, we name this algorithm GSSC (Global Service Selection and Composition). It does not presuppose a restrictive meta model, such as SAWSDL, OWL-S, or WSMO. It however understands OWL and prepares heterogeneous semantic requirements and service descriptions for subsequent similarity analysis and composition. To compare the descriptions, we introduce a non-linear classification and matching approach, the LRAAM, as detailed in Chapter 6. The LRAAM shall compensate for the heterogeneity brought about by the lack of a restrictive meta model (other than OWL itself). It also serves as a distributed, "inner" semantic representation, also called *micro-semantic* (Blank *et al.* 92).

4.1 Building blocks

Initially, free semantic descriptions are produced of business activities and how they are related, and of independent – competing – services (e.g., data model and functionality). Consequently, in a world with lots of requirements and third-party services offerings, many independent ontologies may exist. Heterogeneity is inherent and to be expected. Finding similarity among those ontologies is thus necessary to know which service can support a given business activity. GSSC focuses on those ontologies.

Specifically, similarity is measured among (1) *process ontologies*, describing business activities and precedence rules, and (2) *service ontologies*, describing functionality and data model offered (Ludolph *et al.* 11). Similar to WSMO, our assumption is first, that service and business activity descriptions can be more or less similar, and second, that similarity is a selection criterium. The most similar service (out of many) for a certain business activity is thus selected[1].

4.1.1 Process ontology

The process ontology is an ontological representation for causally related activities. They form a chain of business requirements. Within it, any pair of consecutive activities is identifiable. In principle, there is no restriction as to how domain experts (or ontology engineers) of organizations describe the activities. In Figure 4.1, an example is given[2]. It is aligned to our running example.

Completeness of the representation depends on the organization's domain experts who encode them. Within the figure, some concepts detailing the activity *CreditChecking* are emphasized (cf., magnifying glass). Clearly, further concepts pertaining to IOPEs, goals, data types, etc. could be defined here.

4.1.2 Service ontologies

A service ontology describes the functionality supported by the service (i.e., application) and the data structure, resp. IOPEs, needed for accessing that functionality. In this context, we refer to functionality as the set of functions offered to the user. More than one service might exist

[1] The concept "similarity" is introduced in greater detail in Section 6.3.

[2] More complex process elements, such as "split," "join," etc. are not relevant to show the concept.

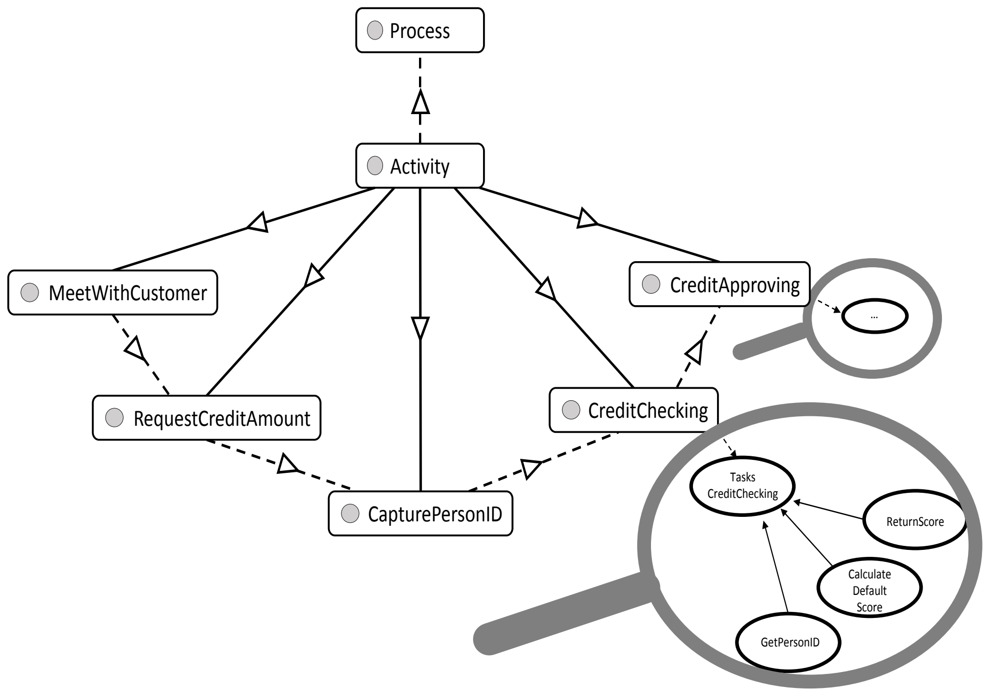


Figure 4.1: Simple ontology for related business activities.

to support a certain activity. Consequently, more than one service ontology might exist geared towards one activity. A simple example for a *CreditChecking* service is given in Figure 4.2.

In light of to the running example, let us assume that a second service is available. It is provided by another Cloud-based service providers. From a requester's perspective, it offers the same functionality as the first credit checking service. The input is *personID*, the output is *score*. However, there is a difference in the quality of the scoring value. It is more reliable due to a more sophisticated risk evaluation algorithm (cf., Fig. 4.3). The second service's ontology reflects this difference.

4.1.3 Sequence ontology

The sequence ontology is an auxiliary ontology (cf., Fig. 4.4). It represents a generic activity sequence. It has three objectives. First, it is used to construct a *reference ontology* describing a pair of related business specifications taken from the process ontology. Second, it is used to construct a *compound ontology* describing how services may implement the reference ontology. The compound ontology is constructed by pairing available services ontologies. Third,

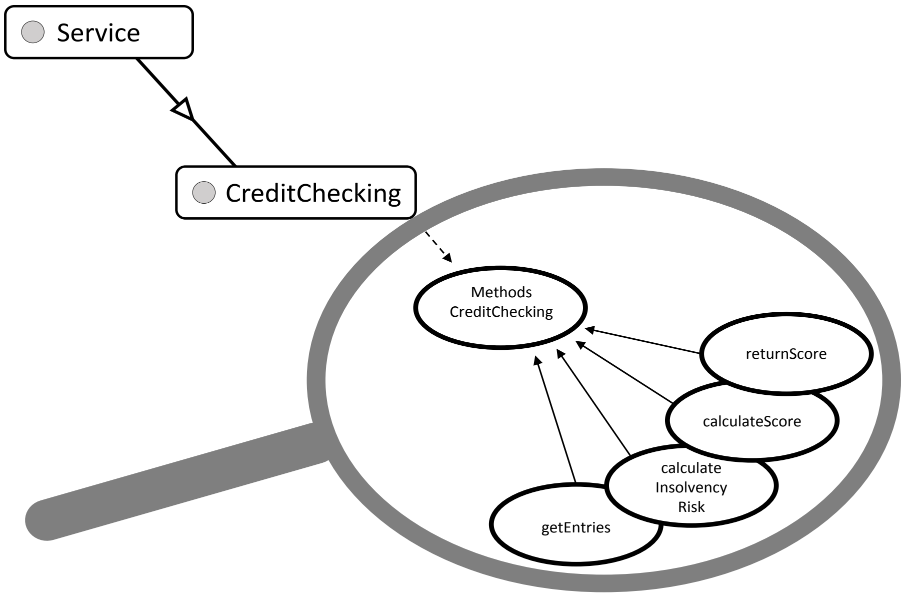


Figure 4.2: Simple ontology for a credit checking service.

the sequence ontology's elements represent the reduced representation for similarity analysis (detailed in Chap. 7).

Specifically, key concepts such as *SourceFunction* and *DestinationFunction* are used by GSSC to attach respective semantic descriptions (magnifying glass). Similarly, data structures can be attached to *SourceDataObject* or *DestinationDataObject*. For simplicity reasons, this however is not discussed during the reminder of the work.

4.1.4 Reference ontology

A reference ontology models a specific sequence of consecutive business activities. It uses the sequence ontology as basis. Descriptions are taken from each consecutive pair of activities described within the process ontology (cf., Fig. 4.1). They are attached at the location of the sequence ontology's relevant placeholders. GSSC uses key elements to accomplish this task, such as *Activity*, *Service*, *SourceFunction*, and *DestinationFunction*. Figure 4.5 shows an example.

Clearly, a detailed representation within the process ontology is warranted to obtain the best possible reference for comparison with alternative service combinations (cf., next section).

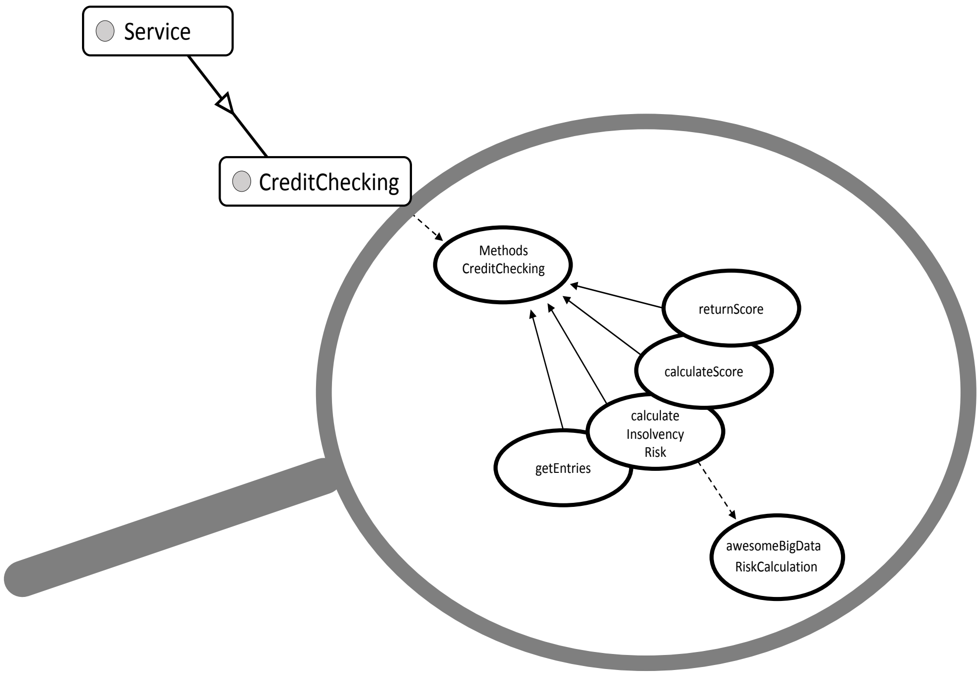


Figure 4.3: Simple ontology for a different credit checking service.

4.1.5 Compound ontologies

A compound ontology is constructed by pairing two available service ontologies. Several service ontologies might exist to support one activity. Therefore, many compound ontologies are constructed and compared against one reference ontology for ontological similarity. Figure 4.6 shows an example of a compound ontology.

Ideally, a more detailed representation of a service improves the likelihood to be selected over a competitive service when alternative service combinations, i.e., compound ontologies, are compared with one reference ontology. The next section sheds more light on this aspect.

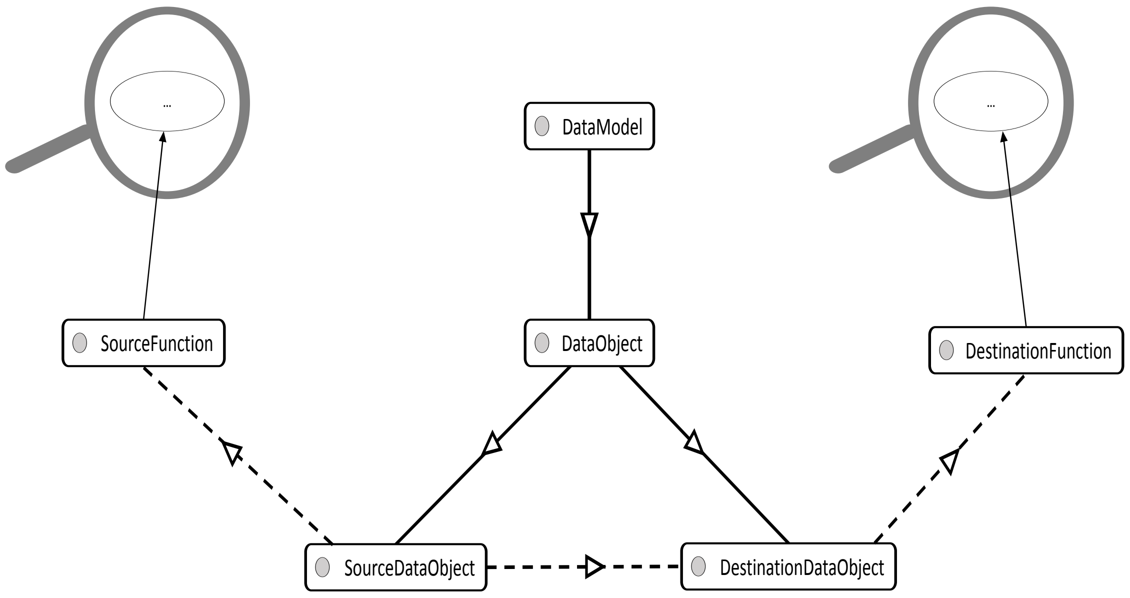


Figure 4.4: Sequence ontology. The placeholders (magnifying glass) are replaced, either with specifications, extracted from the process ontology to construct the reference ontology, or with specification, extracted from the service ontologies to construct the compound ontology.

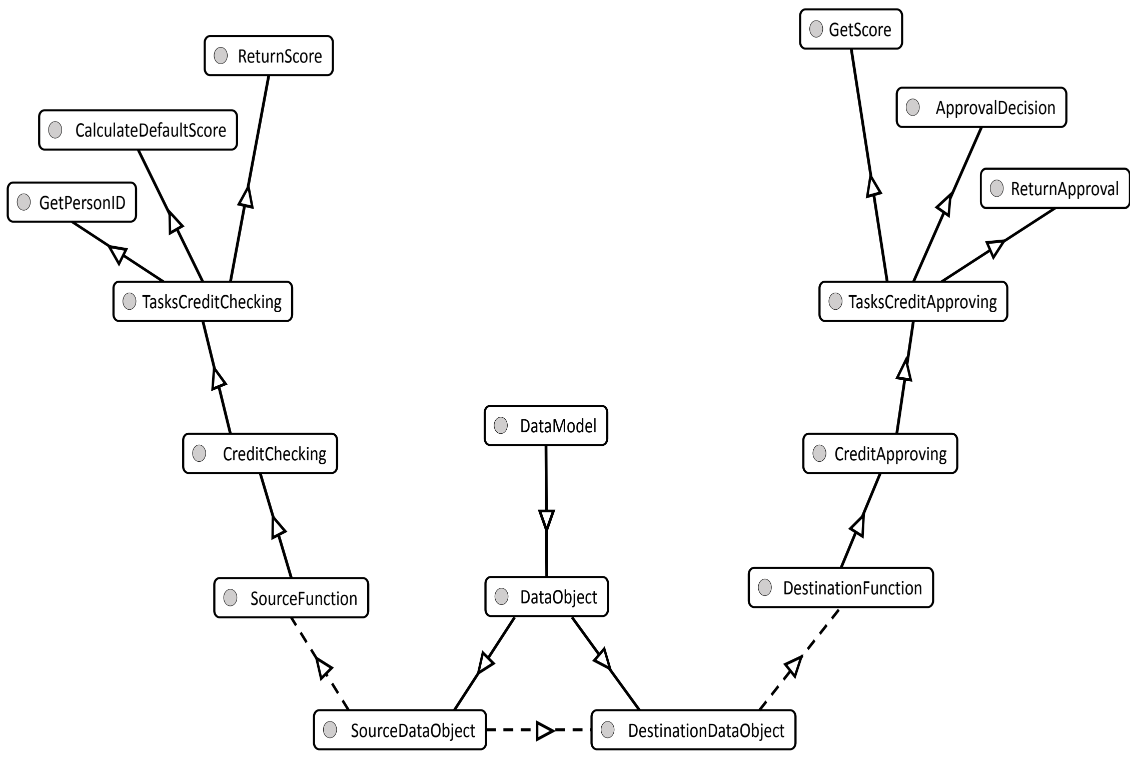


Figure 4.5: Reference ontology with attached semantic description coming from the process ontology. (CreditApproving elements are added for this example.)

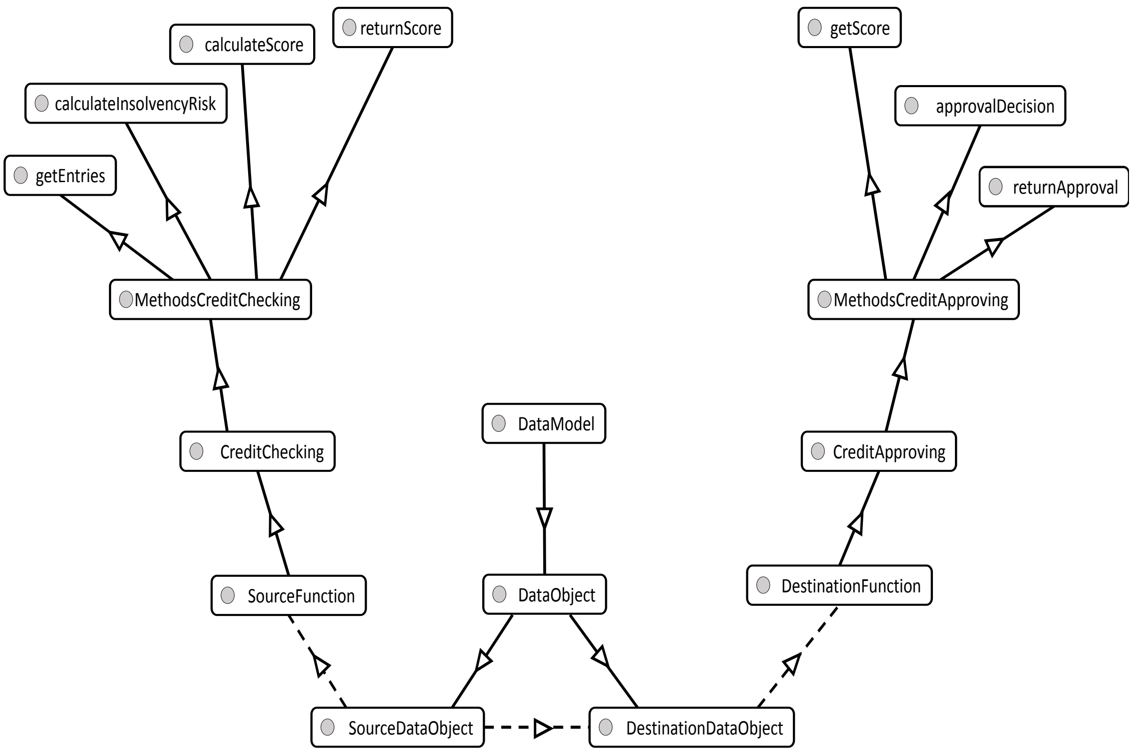


Figure 4.6: Compound ontology with attached semantic description coming from service ontologies. *CreditChecking* elements are different from the respective elements in the reference ontology. (CreditApproving elements remain the same.)

4.1.6 Running example revisited

A specific service combination (*CreditChecking-x-CreditApproving*) is shown in Figure 4.7 as it contrasts with a specific reference ontology. The CreditChecking service is the "normal" one as depicted in Figure 4.2[1]. It used to satisfy the bank's business requirement (cf., Fig. 4.1) for scoring a person's solvency. The dashed frames show the parts relevant for similarity analysis.

Figure 4.8 shows the more sophisticated service (cf., Fig. 4.3) contrasting with the same reference ontology as above. It is reasonable to assume that the ontological distance, notwithstanding the measure, is higher than in the first case shown in Figure 4.7. In other words, the new service is not considered by the bank.

In contrast, Figure 4.9 depicts the more sophisticated service as it contrasts with a **different** reference ontology. The difference could be the result of the bank's changing business requirement. It is now represented within a new ontological description. We might assume that the similarity between the new requirement for *CreditChecking* and the more sophisticated credit-checking service is, henceforward, smaller than between the new requirement and the "normal" service. The new service should thus be selected[2].

Eventually, it ought not to be our task to opine (i.e., guess, assume) which distance is smaller. The system, equipped with a certain degree of "intelligence" should be able to accomplish this.

[1]The CreditApproving service remains the same.

[2]Again, the CreditApproving service is stable and only shown for the sake of completeness in this example.

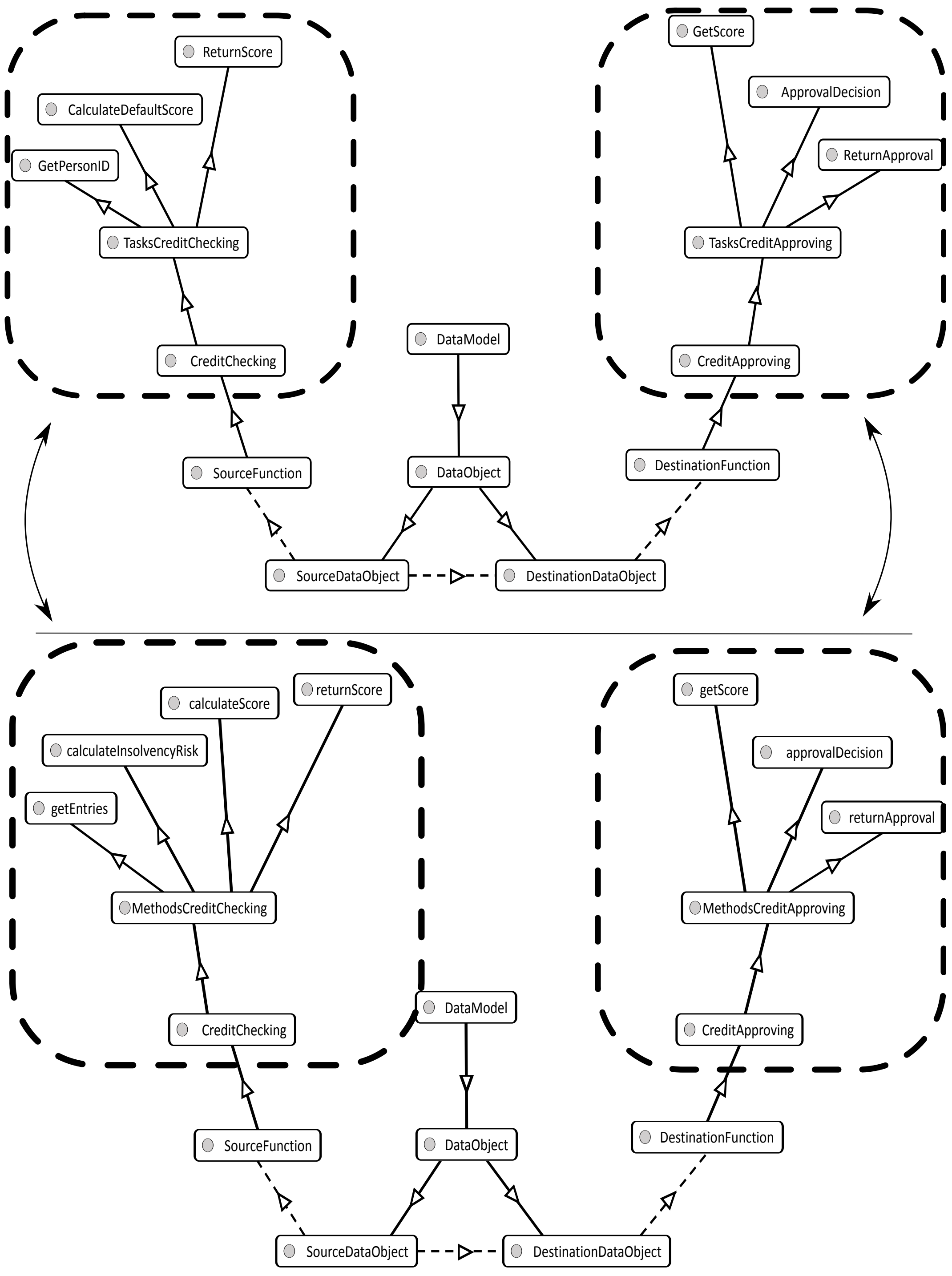


Figure 4.7: Reference ontology (above) and compound ontology (below) for similarity analysis. Dashed frames are relevant for similarity analysis.

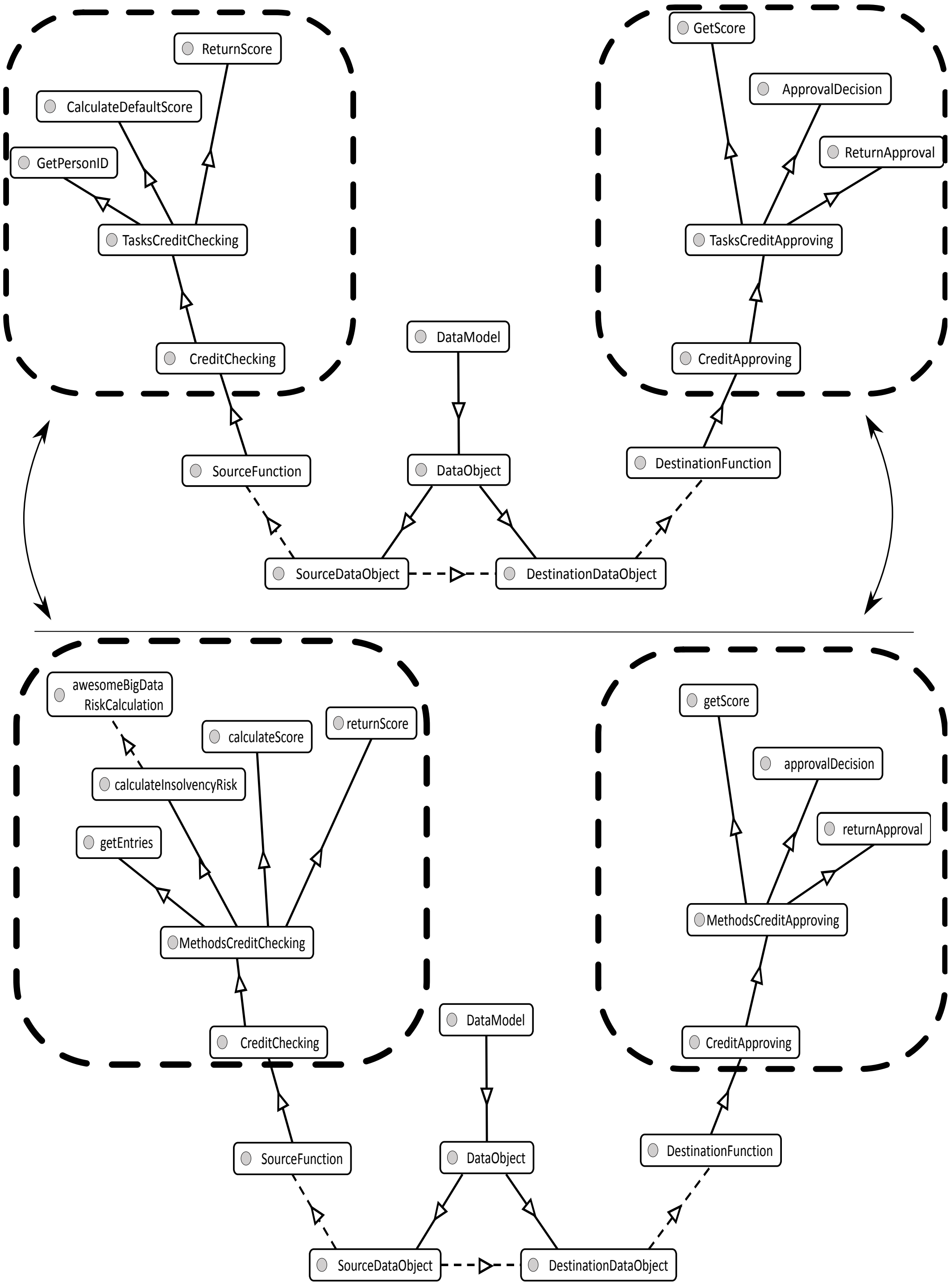


Figure 4.8: Reference ontology (above) and compound ontology for competing service (below).

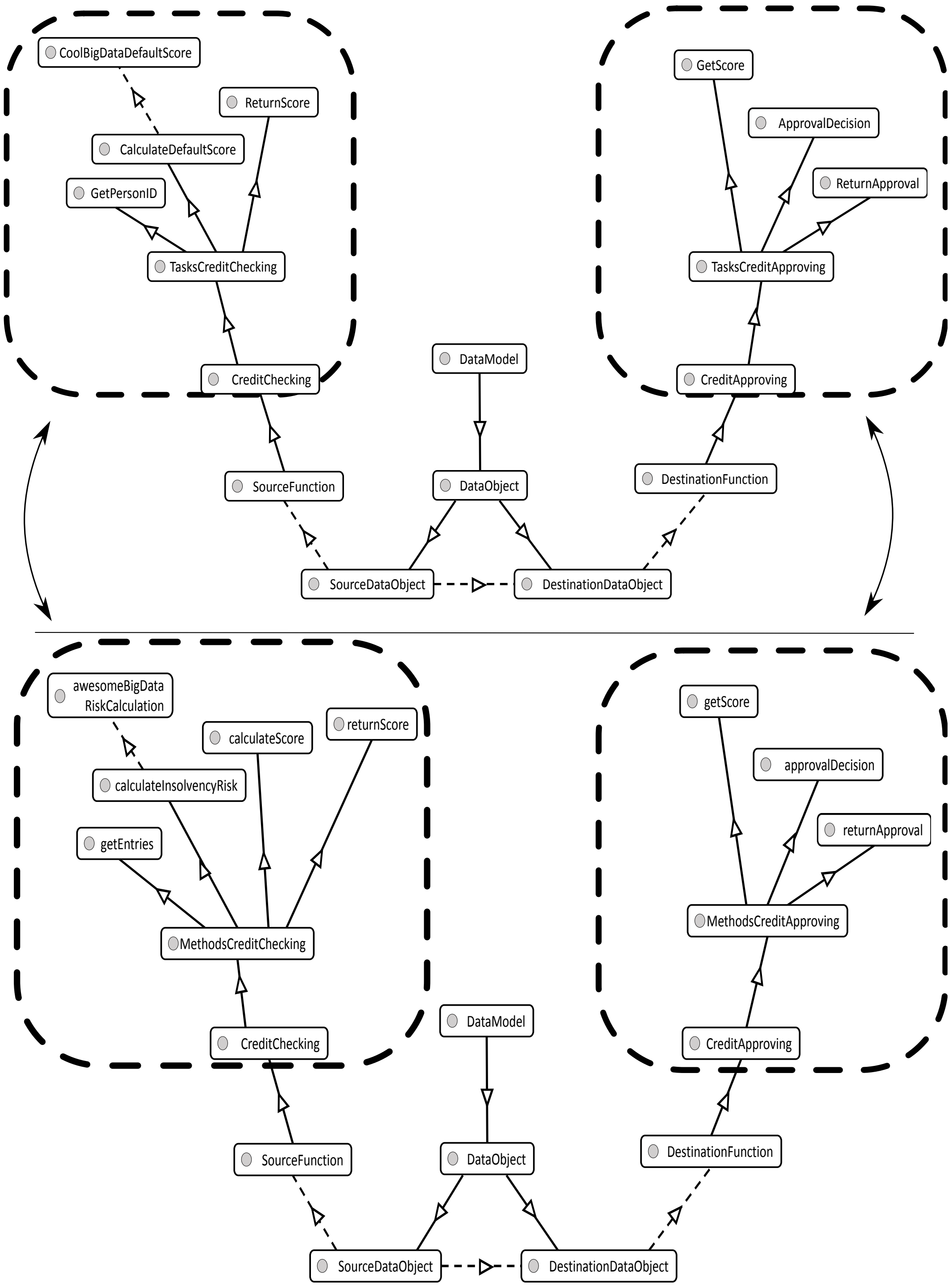


Figure 4.9: Reference ontology for new requirement (above) and compound ontology for competing service (below).

4.2 Global service selection and composition algorithm

After introduction of the building blocks and principles, GSSC[1] is detailed as followed: From a well-defined process ontology, all pairs of consecutive activities are first identified and transformed into a set $R = \{r_{aa'}\}$ of reference ontologies, where $r_{aa'}$ is the reference ontology between activities a and a' (cf., Algo. 4.2.2, *buildReferenceOntologyList*). Furthermore, service ontologies are paired to form a set $C = \{c_{ss'}\}$ of compound ontologies representing services s and s' combined, such that $s \neq s'$ (cf., Algo. 4.2.3, *buildCompoundOntologyList*).

Both sets of ontologies are compared with each other to evaluate ontological similarity. To this end, what we infer to as ontological distance d_{rc} is measured between each pair of reference and compound ontologies. It results in a matrix $D = [d_{rc}]$, where d_{rc} is the ontological distance between reference ontology r and compound ontology c (cf., Algo. 4.2.4, *computeDistanceMatrix*). In Chapter 6, we define the specific measure for d_{rc}. In Chapter 7, we present the respective algorithm to compute d_{rc}.

The distance matrix D is then used by GSSC, which extracts those services (i.e, c) optimally supporting each defined business activity (i.e., r; cf., Algo 4.2.5, *optimizeMapping*). It is achieved by a branch an bound approach minimizing over the computed ontological distances d_{rc}, but also integration and acquisition costs[2].

4.2.1 GSSC

Algorithm 4.2.1: *GSSC*

```
1 buildReferenceOntologyList                    // Algo. 4.2.2
2 buildCompoundOntologyList                     // Algo. 4.2.3
3 computDistanceMatrix      // Algo. 4.2.4, calling LDIST: Algo. 7.2.1 to Algo. 7.2.10
4 optimizeMapping                               // Algo 4.2.5
```

[1]For readability divided into parts.

[2]For our experiments however, we fixed costs at 0, as the focus at this time is only on d_{rc}.

4.2.2 Build reference ontology list $R = \{r_{aa'}\}$

Algorithm 4.2.2: *buildReferenceOntologyList* – constructs the set of reference ontologies for every consecutive activity pairs

Goal : Create a reference ontology for each activity pair (a, a') such that a precedes a' in the process.

Input : A process ontology showing activities and precedence rules. Activity description ideally includes modeling of concepts manipulated by the activity.

Output : The result is set $R = \{r_{aa'}\}$, the set of reference ontologies, where $r_{aa'}$ is the reference ontology between activities a and a'. The output activity of reference ontology $r_{aa'}$ is a, noted $A^O(r_{aa'})$, whereas the input activity of reference ontology $r_{aa'}$ is a', noted $A^I(r_{aa'})$.

```
1 $R \leftarrow \emptyset$
2 Seek the class *Activity* in the process ontology
3 Let $A$ be the set of classes being domain of the *subClass* relation where *Activity* class is the range
4 **foreach** $a \in A$ **do**
5     **foreach** *FollowedBy* property $p$ for which $a$ is domain **do**
6         **foreach** class $a'$ range of $p$ **do**
7             Construct $r_{aa'}$ from the sequence ontology such that $a$ is attached as *subClass* of SourceService and $a'$ is attached as *subClass* of DestinationService, including modeling concepts manipulated by the activity.
8             $R \leftarrow R \cup \{r_{aa'}\}$
```

4.2.3 Build compound ontology list $C = \{c_{ss'}\}$

Algorithm 4.2.3: *buildCompoundOntologyList* – Creation of compound ontologies

Goal : Create all compound ontologies for all distinct services sets that can be performed by the available applications.

Input : The set of application ontologies that describe the set of all services in an application and the data model used by these services either as input or as output, or both.

Output : The result is a set $C = \{c_{SS'}\}$, the set of compound ontologies for sets of services S and S', such that $S \cap S' = \emptyset$. The output service set of compound ontology $c_{SS'}$ is S, noted $S^O(c_{SS'})$, whereas the input service set of compound ontology $c_{SS'}$ is S', noted $S^I(c_{SS'})$.

```
1  $C \leftarrow \emptyset$
2  $\mathbf{S} \leftarrow \emptyset$
3  foreach application ontology $o$ do
4      Seek the class Service in $o$
5      $\mathbf{S} \leftarrow \mathbf{S} \cup \{s|s$ is a class in $o$ and $s$ is a subClass of Service and $s$ is range of
       $linkDataService\}$
6  foreach $S \in \mathcal{P}(\mathbf{S}) \setminus \{\emptyset\}$                    // $\mathcal{P}(\mathbf{S})$ is the power set of $\mathbf{S}$
7  do
8      foreach $S' \in \mathcal{P}(\mathbf{S}) \setminus \{\emptyset\}$ such that $S \cap S' = \emptyset$ do
9          Construct $c_{SS'}$ from the sequence ontology such that all services in $S$ are
           attached as subClass of SourceService and all services in $S'$ are attached as
           subClass of DestinationService, including data modeling manipulated by
           the services.
10         $C \leftarrow C \cup \{c_{SS'}\}$
```

4.2.4 Determine ontological distance matrix $D = [d_{rc}]$

Algorithm 4.2.4: *computeDistanceMatrix* – Construct distance matrix between reference and compound ontologies

Goal : Determine ontological distance between each pair of reference and compound ontologies.

Input :

- $R = \{r_{aa'}\}$, the set of reference ontologies, where $r_{aa'}$ is the reference ontology between activities a and a'.
- $C = \{c_{SS'}\}$, the set of compound ontologies for sets of services S and S', such that $S \cap S' = \emptyset$.

Output : The result is matrix $D = [d_{rc}]$, where d_{rc} is the ontological distance between reference ontology r and compound ontology c.

```
1 $C \leftarrow \emptyset$
2 S $\leftarrow \emptyset$
3 foreach $r \in R$ do
4     foreach $c \in C$ do
5         $d_{rc} \leftarrow$ distance between reference and compound ontology. Different
          distance functions can be tested in order to choose the most efficient // e.g.,
          the LRAAM-based function as described in Chapters 6 and 7.
```

Step 5 within Algorithm 4.2.4 serves as a place holder. Various matching methods can be used to compute d_{rc}. A brief overview about these is given in Chapter 5. Thereafter, we explore the LRAAM-based method (cf., Chap. 6 and 7, specifically Algo. 7.2.10, *compute distance* d_{rc}).

With the above distance matrix, the subsequent (main) algorithm calculates the optimal set of compound ontologies to support all consecutive reference ontologies.

4.2.5 Determine optimal set of compound ontologies

The distance matrix provides individual distances among all compound and reference ontologies. In a next step, a branch and bound optimization algorithm (cf., Algo 4.2.5, *optimizeMappingff.*) determines the optimal set of compound ontologies to support all reference ontologies. It receives input defined as follows:

- Reference ontology precedence matrix

  with $a^i(r)$ input activity of r, and $a^o(r')$ output activity of r'

  $$\mathfrak{N} = [\mathfrak{n}_{rr'}] \text{ where } \begin{cases} 1 & \text{iff } a^i(r) = a^o(r') \\ 0 & \text{otherwise} \end{cases}$$

  It enforces the sequence of activities as defined by the business expert.

- Compound Ontology Sequence Matrix

  with $s^i(c)$ input service of c, and $s^o(c')$ output service of c'

  $$\mathfrak{K} = [\mathfrak{k}_{cc'}] \text{ where } \begin{cases} 1 & \text{iff } s^i(c) = s^o(c') \\ 0 & \text{otherwise} \end{cases}$$

  It enforces a sequence of adequate services to support a sequence of activities, ideally the above one.

- Service - Compound Ontology Membership Matrix

  $$\mathfrak{M} = [\mathfrak{m}_{cs}] \text{ where } \begin{cases} 1 & \text{iff } \exists s^o(c) \text{ or } \exists s^i(c) \\ 0 & \text{otherwise} \end{cases}$$

  It assures that a service is part of a compound ontology, either to produce an output or to receive an input.

- Decision Matrix

  $$\mathfrak{X} = [\mathfrak{x}_{rc}] \text{ where } \begin{cases} 1 & \text{when } c \text{ is matched with } r \\ 0 & \text{otherwise} \end{cases}$$

  It indicates matching of a compound ontology to a reference ontology.

- Set of available on-premise applications $P = \{p\}$

- Application purchase cost ρ_p is the cost to acquire application p containing a set of services $S_p = \{s_p\}$.
- Off-premise service purchase cost ρ_s is the cost to acquire/use/consume service s either to produce an output or to supply it an input.
- κ is the linear unit cost to integrate non-integrated functionality of s or p, that is, whenever $d_{rc} \neq 0$, the cost to reconcile differences between functionality and activity by means of an integration effort.

Given the definitions above, the objective of the optimization algorithm is to find

$$\min \left(\sum_{r \in R} \sum_{c \in C} \mathfrak{x}_{rc} \cdot \left(\kappa \cdot d_{rc} + \sum_{p \in P} \min \left(1, \sum_{s \in S_p} \mathfrak{m}_{cs} \right) \cdot \rho_p + \sum_{s \in \mathbf{S}} \mathfrak{m}_{cs} \cdot \rho_s \right) \right) \tag{4.1}$$

In words, the focus is on (1) identifying matches among R and C ($\mathfrak{x}_{rc}$), which minimize (2) integration costs (κ), (3) costs of on-premise applications packages/add-on's providing services bundles (ρ_p), and (4) costs of off-premise, cloud-based services (ρ_s).

The optimization is subject to the following constraints:

1. Pruning for optimality: each r must be matched with exactly one c.

$$\sum_{c \in C} \mathfrak{x}_{rc} = 1, \forall r \in R$$

2. Pruning by upper bound: at maximum one c is matched with an r.

$$\sum_{r \in R} \mathfrak{x}_{rc} \leq 1, \forall c \in C$$

3. Pruning by infeasibility

$$\mathfrak{n}_{rr'} \cdot \mathfrak{x}_{rc} \cdot \mathfrak{x}_{r'c'} \leq \mathfrak{k}_{cc'}, \forall r, r' \in R, \forall c, c' \in C$$

Mapping optimization

Algorithm 4.2.5: *optimizeMapping*

```
1 $x[]$ ← new array of compound ontologies of $|R|$ elements. // Note that the $i^{th}$ element of $x$ is the compound ontology matched to the $i^{th}$ member of the set of reference ontologies $R$. We will refer to the $i^{th}$ member of the set of reference ontologies $R$ as $R[i]$.
2 $xOpt[]$ ← new array of compound ontologies of $|R|$ elements.
3 $cost \leftarrow 0$
4 $optCost \leftarrow \infty$
5 for $c \in C$ do
6     extendSolution($x$, $cost$, $xOpt$, $optCost$, 0, $c$)
```

Algorithm 4.2.6: *extendSolution*(x, $cost$, $xOpt$, $optCost$, $level$, $choice$)

```
1 $x[level] \leftarrow choice$
2 $cost \leftarrow cost$+computeResidualCost($level$, $choice$)
3 if partialSolutionOK(x, cost, optCost, level) then
4     if $level = |R| - 1$ then
5         substituteOptimal($x$, $cost$, $xOpt$, $optCost$)
6     else
7         for $c \in C$ do
8             extendSolution($x$, $cost$, $xOpt$, $optCost$, $level + 1$, $c$)
9 $x[level] \leftarrow null$
```

Algorithm 4.2.7: *computeResidualCost*($level$, $choice$)

```
$residual \leftarrow \kappa \cdot d_{R[level],choice}$
for $p \in P$ do
    $min \leftarrow 0$
    for $s \in S_p$ do
        $min \leftarrow min + \mathfrak{m}_{choice,s}$
    if $min \geq 1$ then
        $residual \leftarrow residual + \rho_p$
for $s \in$ S do
    $residual \leftarrow residual + \mathfrak{m}_{choice,s} \cdot \rho_s$
return $residual$
```

Algorithm 4.2.8: *partialSolutionOK*(x, $cost$, $optCost$, $level$)

```
if $optCost < cost$ then
    return false                                // optimal constraint
for $i$ from 0 to $level - 1$ do
    if $x[i] = x[level]$ then
        return false                            // 2nd constraint
    if $\mathfrak{n}_{R[i],R[level]} > \mathfrak{t}_{x[i],x[level]}$ then
        return false                            // 3rd constraint
return true
```

Algorithm 4.2.9: *substituteOptimal*(x, $cost$, $xOpt$, $optCost$), $level$)

```
if $optCost > cost$ then
    for $i$ from 0 to (size of $x$)$-1$ do
        $xOpt[i] \leftarrow x[i]$
    $optCost \leftarrow cost$
```

4.3 Discussing the global solution

Under the conditions described above, it can be inferred that for the distance measure (i.e., dissimilarity function) $d_{rc} \in \{0, \ldots, u\}$, with $u \in \mathbb{R}, \forall r \in R, \forall c \in C$, the perfect value would be $d_{rc} = 0$[1]. It would return an optimal, integrated sequence of services to support a predefined business process. Integration costs would be non-existent. Additionally, it would merely become a question of non-functional aspects, namely of application and service purchasing costs.

However, this is an idealistic case. If nothing else, in the unforeseeable dynamic service landscape, services are even expected to be different from business requirements. Service offerings ought to be standardized in order to be reusable. Business requirements ought to be dynamic, as they represent a company's differentiation efforts.

The challenge is thus to find a matching method to *minimize* d_{rc}, which would respectively *minimize* the objective function 4.1.

As stated at the beginning of this chapter, pure symbolic matching approaches are limited in accomplishing this task. Strict symbol systems, i.e., meta models, need to be established. Comparison and matching is then based on shapes which have nothing to do with true meaning. This, eventually, limits flexible integration automation.

In the next chapter, we briefly describe some ontology matching methods before we introduce the non-linear classification method, the LRAAM. We explore the latter as means to cope with the aforementioned shortcomings of symbolic matching methods.

[1]cf., Sect 6.3ff. for a detailed definition of d.

5

Ontology-based matching

In the preceding chapter, we argued that ontologies shall be used to steer the service integration, i.e., composition, process. Thereby, our reasoning entailed the existence of a variety of *a priori* heterogeneous ontologies. They may contain different conceptualizations stemming from different domains.

In such a scenario however, we expect to find intersections of the semantic descriptions (and by extension of the service functionality to be integrated). Without such a relation an integration attempt would be useless. Clearly, a credit checking service is semantically more similar to a credit approval service (e.g., with the concepts *score* or *personID*) than it is to a weather forecast service (e.g., without the concepts *score* or *personID*). In short, ontologies need to be related or *matched* in a fashion to consistently derive the best possible service composition. Once a best match is found, it can be used to formulate instructions for the specific service integration (such as presented in Chap. 3).

Ontology matching as discipline aims at proposing mechanisms to find semantic similarities. As described in Figure 3.8, they usually stand for equivalence, super relation, subsumption, or disjointness between ontological entities. The results, called *alignments*, express with various degrees of (symbol-based) precision the relation between the ontologies (Euzenat and Shvaiko 07).

5.1 Ontology matching and alignment

A *match* is the fundamental operation to identify similarity (Rahm and Bernstein 01). It takes two ontologies as input and produces a mapping between semantically similar entities.

Following (Kotis *et al.* 06), the matching of two ontologies o and o' can be defined as a morphism from o to o'. The associated matching task is to find an alignment A between o and o' (Euzenat and Shvaiko 07). An alignment is then a set of correspondences, which in turn is a 4-tuple $\langle id, e_o, e_{o'}, r\rangle$, with id as correspondence identifier, e_o and $e_{o'}$ as entities (e.g., classes, properties) of the compared ontologies, and $r \in \{\leqslant, =, \geqslant, \perp\}$[1] the identified relation (Atencia *et al.* 11).

Furthermore, instead of aligning two ontologies "directly," intermediate foundational, domain, or core reference ontologies (e.g., WordNet, DBPedia, etc.) can be used. They serve as a common denominator for term and axiomatic reference. Given two source ontologies o and o', the merging problem is then to find an alignment by mapping them to the intermediate ontology and obtain the minimal union of their terms with respect to the their alignment (Kotis *et al.* 06).

5.2 Application areas for ontology matching

In general, matching is a typical operation where heterogeneous structural models are encountered, such as for database schemas, data centers, supply chains, etc. As stated at the beginning of Chapter 4, without semantic annotations, those matching operations are mostly done manually or semi-automatically. With the advent of dynamic network structures and data flows (e.g., driven by Cloud-based service offerings, agents, or peer-to-peer systems), runtime matching will likely be required (Euzenat and Shvaiko 13). The authors thus advocate to replace manual approaches by automatic matching techniques (cf., next section). For example, in Table 5.1, they suggest application areas and the matching subject.

5.3 Ontology matching techniques

Various techniques are used to find correspondences. Following (Euzenat and Shvaiko 07, Shvaiko and Euzenat 05, Euzenat and Shvaiko 13), they are classified into syntactic and (formal) semantic matching techniques (cf., Table 5.2).

[1]reads: is less general than, equal, is more general than, disjoint from.

Table 5.1: Areas for ontology matching

Application	**Matching subject**
Ontology engineering	transformation
Schema integration	merging
Catalogue integration	data translation
Data integration	query mediation
P2P information sharing	query mediation
Web service composition	data mediation
Multi agent communication	data translation
Context matching in ambient computing	data translation
Semantic web browsing	navigation
Query answering	query reformulation

Table 5.2: Classification of matching techniques (adapted from (Euzenat and Shvaiko 13)).

	Syntactic	**Semantic**
Element-level *(considers classes or their instances in isolation from their relations with other classes or their instances)*	**Informal resource-based** (*directories, annotated resources*): relates ontologies to informal resources, e.g., annotating encyclopedia pages or pictures. For example, two classes annotating the same set of pictures are considered equivalent. Techniques exploit data analysis and statistical approaches to find regularities (cf., below). **String-based** (*name similarity, description similarity, global namespace*): the more similar strings are, the more likely they denote the same concept. A distance functions maps a pair of strings to a real number. A smaller number indicates a greater similarity, such as the Hamming distance described in Section 4.2.	**Formal resource-based** (*upper-level ontologies, domain-specific ontologies, linked data*): uses external ontologies to perform matching. Specifically, alignments among local ontologies are facilitated by one or more referential ontologies.

– continued on next page

Table 5.2 (continued from previous page)

	Syntactic	**Semantic**
	Language-based (*tokenisation, lemmatisation, morphology, elimination, lexicons, thesauri*): based on natural language, exploiting morphological properties of words, e.g., tokenization: $Hands - Free\_Kits \rightarrow < hands, free, kits >$, lemmatization: $Kits \rightarrow Kit$ **Constraint-based** (*type similarity, key properties*): deals with the internal constraints being applied to the definitions of entities, such as types, cardinality of attributes, and keys. Entities with comparable internal structure or properties with similar range and domain are used to create correspondence clusters.	
Structure-level *(considers concepts or their instances to compare their relations with other concepts or their instances)*	**Taxonomy-based** (*taxonomy structure*): graph algorithms, which consider only specialization relations. A is-a link is supposed to connect terms which are already similar (subsets, supersets of each other). Neighbors might therefore also be similar. **Graph-based** (*graph homomorphism, path, children, leaves*): graph algorithms which consider input ontologies as labeled graphs. Similarity is based on the analysis of the terms' position within the graphs.	**Model-based** (*SAT solvers, DL reasoners*): algorithms that handle the input based on its semantic interpretation, e.g., model-theoretic semantics. If two entities are the same, they share the same interpretation. They are well-grounded in deductive methods (cf., Sec. 3.1).

– continued on next page

Table 5.2 (continued from previous page)

	Syntactic	Semantic
	Instance-based (*data analysis, statistics*): compare instances (i.e., objects) of classes to decide if classes match. They may rely on data analysis (e.g., correspondence analysis) and statistical techniques (e.g., frequency distributions).	

These matching techniques differ in respect to speed, resource consumption, degree of precision, completeness and thus in their suitability concerning the different application areas (cf., Sect. 5.2).

5.4 Discussing the gap

Let us assume a single consistent global ontology. It could be used to uniquely describe any service conceivable now and in the future. In such a scenario, symbolic matching methods would provide suitable candidates for finding, at least theoretically, perfect matches, i.e., $A = R$. The reason is that A as well as R would be conceived from the same "vocabulary." Applied to our context, it could for the distance measure $d_{rc} \in \{0, \ldots, u\}$ effectively return a value of $d_{rc} = 0$. In the following points, however, we depict more or less obvious concerns with this assumption and the above described matching in general:

- As stated in Section 3.4, in reality, there is no such a semantic umbrella, i.e., global ontology. It is thus not even likely that an A exists to completely correspond to R. In fact, we do not know the future, different languages (yet to be), or how independent organisations define the world, or else, services and requirements, in different terms.

- R is a reference provided by a human expert. Clearly, it is problematic in the sense of understanding *unknown* conceptual input towards meaningful actions (as mentioned at beginning of Chapter 4).

- Matching techniques mentioned above focus on individual elements or whole structures. They analyze frequency distribution or specific languages' morphologies. They introduce external data repositories. Eventually, they all work on discrete arbitrary objects, that is, symbolic signifiers. Eventually, they rely on the exactness of those in order to execute the appropriate, yet predefined, operations. The introduced quality measures are also submitted to this principle.

- No inner representation of the *signified* exists. To the symbolic matching system, actually, the signified does not even exist. Therefore, the above techniques do not operate when confronted with undefined *signifiers* (imagine the Chinese room experiment, depicted in Section 3.1, this time with syntactic rules to manipulate the Chinese symbols not written in English, but for example in Chinese.)

All points are at odds with the undeniable fact of continuous variations and lack of presupposed exactness in the problem domain, namely the ever shifting reality of organizations.

In the next chapter, the described symbol-based approach is thus combined with a non-deterministic approach. Specifically, we depict an artificial neural network (ANN) implemented as LRAAM (Labelled Recursive Auto-Associative Memory). As opposed to the above symbolic focus, it concentrates on parallel data processing and distributed representations. Such distributed entities can be called *subymbolic* (Blank *et al.* 92), or else, *sublexical symbols* (Chan 03). In the next two chapters, we detail their integration into the overall approach. We also depict why this may be beneficial to alleviate the shortcoming of purely symbolic approach.

6

LRAAM-based matching

The Labelled Recursive Auto-Associative Memory (LRAAM) is an artificial neural network (ANN) implementation with a specific network topology. It is able to transform complete graph structures into distributed representations. In this chapter, we introduce and explore it as a matching method to compare semantic, graph-like, descriptions. Our objective is to better account for variations in ontological descriptions, which otherwise may be incorrectly or not at all classified by symbol-based matching methods. We obtain the distance measure d_{rc}, which is used by GSSC presented in Chapter 4 to determine appropriate services to support requirements.

In the following sections, the specific LRAAM topology is introduced. We explain how it shall help to calculate the distance d_{rc} between semantic descriptions, namely reference and compound ontology. We also discuss the approach in light of the running example.

6.1 Artificial neural networks

Typical symbols in a symbolic model are letter strings, such as an ontological class (e.g., person). A symbol may be placed into structured relationships with other symbols, e.g., subsumption (cf., Fig. 3.8). It might be bound to a variety of values during the course of processing. However, it never changes its (arbitrary) shape. As discussed in Section 3.1, one can replace the shape by A, B, etc. and still keep its formal-semantic position within a logical defined system of axioms. The label is simply an atomic label possessing no internal structure of its own (Blank *et al.* 92).

As opposed to logics- and symbol-based concepts, an ANN belongs to the *subsymbolic* or *distribute* paradigm. It represents distributed patterns, i.e, subsymbols, of continuous values. During processing, these patterns might evolve into slightly different patterns. They nevertheless still behave in a closely related way to the original pattern. Thus, specific values within subsymbolic patterns cannot be replaced by arbitrary other values and still produce the same results.

ANNs, and specifically the LRAAM implementation, have the ability to extract patterns from complicated or changing input data. In our context, this is used to evaluate similarity of ontological patterns. Specifically, it pertains to changing r and c. These changes are supposed to be steady and can thus be *too difficult* (i.e., too subtle) to be noticed by either humans or symbol-based computer techniques (Li *et al.* 12).

In (Blank *et al.* 92), three major pillars distinguish the subsymbolic from the symbolic representations: the type of representation, the style of composition, and the functional characteristics. They are summarized in Table 6.1.

Table 6.1: Symbolic vs. subsymbolic paradigm

	Subsymbolic	**Symbolic**
Representation	distributed	atomic
	continuous	discrete
	emergent	static
	use affects form	arbitrary
Composition	superimposed	concatenated
	context-sensitive	systematic
Functionality	micro-semantic	macro-semantic
	holistic	atomic

ANN's interesting properties do not arise from the units' individual functionality but from the collective effects resulting from the interconnections of these units. The network is able to encode input patterns into intermediate patterns of activation spread across the hidden layer. It thus amounts to the development of distributed *internal* representations of the input information (Blank *et al.* 92).

Since our research topic conceptualizes this input information as being symbolic, the basic rationale of a concerted, symbolic/connectionist approach is established. In the next section, the distributed, i.e., the LRAAM, and symbolic, i.e., ontological, paradigms are thus brought together.

6.2 LRAAM as matching method

The LRAAM is a particular implementation of an ANN. Its most intriguing feature is the potential to encode (and decode) labeled directed graphs of arbitrary size (de Gerlachey *et al.* 94, Sperduti 93). The resulting patterns are sensitive to the graph they represent. Following (Ellingsen 97), these patterns can be exploited for *similarity* analysis. They are thus used to calculate the distance measure d_{rc}.

Architecture

The general architecture of an LRAAM is shown in Figure 6.1. It is a supervised 3-layer feed-forward network trained by backpropagation. The term “auto-association” is due to the equality of input and output layers. The dashed arrows indicate that this auto-associative architecture must be used recursively (Pollack 90). Certain node values from the hidden and output layer are fed back to the input data until the network has reached a steady state.

This is achieved by training the network through backpropagation so it learns an identity function $F : \mathbf{x} \rightarrow \mathbf{x}'$, where $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^n$. A node vector is compressed by using the function $F_c : \mathbf{x} \rightarrow \mathbf{z}$. Then, the compressed representation is reconstructed using the function $F_r : \mathbf{z} \rightarrow \mathbf{x}$. The node vector $\mathbf{x}'$ is thus an approximated output equal to $\mathbf{x}$. The network is trained by presenting the input vectors repeatedly, one vector at the time. In the next section, we depict how we use F to measure similarity between r and c, expressed by d_{rc}.

6.3 From symbolic to subsymbolic similarity

In Section 1.3.1, we stated that service and business activity descriptions can be more or less similar, and that similarity is a selection criterium. In Section 2.5, we furthermore explained the *long-term zero-profit equilibrium* phenomenon as reason for a likely proliferating number of competing services with similar functionality. In Chapter 3, we presented semantic techniques which rely on syntactic and formal-semantic similarity of concepts for service discovery, for example based on input or goal definitions. In Chapter 5, we introduced matching techniques

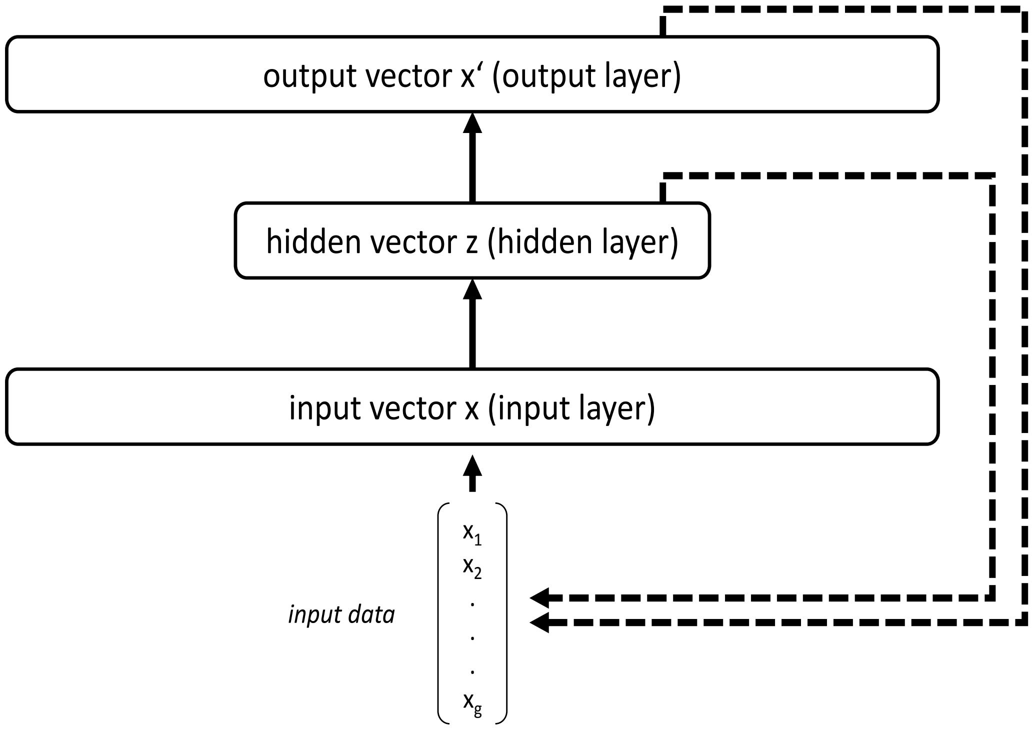

output vector x' (output layer)
hidden vector z (hidden layer)
input vector x (input layer)
input data
x_1
x_2
x_g

Figure 6.1: General LRAAM network architecture.

used to find similar concepts. Similarity is thus a fundamental concept to us. Its application to our context is depicted in more detail in what follows.

General aspects

In (Lin 98), similarity is approached as follows:

- Similarity between a thing A and a thing B is related to their commonality. The more commonality they share, the more similar they are.
- Similarity between a thing A and a thing B is related to the differences between them. The more differences they have, the more dissimilar they are.
- Maximum similarity between a thing A and a thing B is reached when A and B are identical, no matter how much commonality they share.

It is aligned to the notion of dissimilarity and distance given in (Euzenat and Shvaiko 13), where for a set o of entities, a dissimilarity $\delta : o \times o$ is a function from a pair of entities, such that

$$\forall x, y \in o, \delta(x, y) \geq 0 \text{ (positiveness or dissimilarity)}$$
$$\forall x \in o, \delta(x, x) = 0 \text{ (minimality or maximum similarity)}$$

A distance (metric) $\delta : o \times o$ is then a dissimilarity function satisfying the following conditions:

$$\forall x, y \in o, \delta(x, y) = 0 \text{ iff } x = y \text{ (definiteness or maximum similarity)}$$
$$\forall x, y, z \in o, \delta(x, y) + \delta(y, z) \geq \delta(x, z) \text{ (triangular inequality or dissimilarity)}$$

From conceptual (dis)similarity to an LRAAM-based distance measure

Ontological descriptions of business requirements lead to the conceptualization r. Ontological description of services lead to the conceptualization c. Both conceptualizations can be analyzed for similarity (cf., Sect. 3.3).

Specifically, we state that $r = x$, $c = y$, and $r, c \in o = \{R, C\}$. Then, we have a distance measure (or dissimilarity function) $\delta(x, y) \rightarrow d(r, c) \rightarrow d_{rc} = r \times c \in [0, u]$ with $u \in \mathbb{R}$. Thereby, d pertains, first, to the similarity of class or relation labels, and second, also to how classes are related, i.e., to the graph structure of r and c.

First, the similarity of labels (class or relation) could for example be calculated by string-based methods. These methods take advantage of the structure of the symbol string (cf., Hamming distance, Sect. 4.2). They for example would identify class labels, such as *Book* and *Textbook* as more similar than *Book* and *Volume* (Euzenat and Shvaiko 13).

The similarity of how classes are related could (2) be calculated based on the ontological graphs' topologies (cf., Graph-based matching, Table 5.2). For example, the similarity of two graphs can be expressed by a *graph edit distance* (ged). In (Zheng *et al.* 13), it is described as the minimum number of primitive operations (e.g., insert, delete, substitute vertices or edges) needed to transform a graph g_1 to g_1', such that $g_1' = g_2$, denoted by $ged(g_1, g_2)$. Note that ged is a dissimilarity measure, measuring the shortest operation sequence length. In other words, fewer operations result in a smaller distance (metric), and hence in a higher graph similarity. The problem of ged however is that versions of the algorithms are exponential as graphs grow (Koutra *et al.* 11). It is thus not easily applicable to large graphs.

As stated before, with the LRAAM we explore a different approach to similarity. We do not directly operate on symbols or graph topologies. Again, we rather use the fact that the LRAAM learns patterns, which are sensitive to the graph they represent. This sensitivity includes vertex or edge labels *and* the graph topology at the same time.

6.4 Integrating the LRAAM matching method

Each $r \in R$ and $c \in C$ corresponds to a directed acyclic graph $G = (V, E)$. Their entities, such as classes or properties are represented as vertices (V), the connections among them as edges (E) (Eder and Wiggisser 07). The vector $\mathbf{z} \in \mathbb{R}^m$ serves as comparison pattern and $\mathbf{Z} = [\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_{\hat{g}}, \ldots, \mathbf{z}_g]^T$ as a collection of comparison patterns. Thereby, g is the number of vertices in G. Furthermore, $\hat{g}$ is the number of vertices stemming from the sequence ontology, which are by construction present in r and c (cf., Fig. 4.4). Consequently, they correspond to $\mathbf{Z}_r^{\hat{g}}$, respectively $\mathbf{Z}_c^{\hat{g}}$. From Section 6.1, it is known that these collections contain subsymbolic (i.e., micro-semantic) information about the complete collections $\mathbf{Z}_r$ and $\mathbf{Z}_c$, notwithstanding the number of vertices in r or c. Therefore, they can be used to calculate an Euclidean distance among an equal amount of $\mathbf{z}_r$ and $\mathbf{z}_c$, namely $\hat{g}$, corresponding to d_{rc}. The ontological distance between r and c is thus defined as:

$$d_{rc} = \sqrt{\sum_{i=1}^{\hat{g}} (\mathbf{z}_r^i - \mathbf{z}_c^i)^2}$$

The experimental implementation of the LRAAM is further specified in Figure 6.2. Each vertex within G serves as a single input vector $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$. By extension, $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_g]^T$, the collection of vertex vectors for a specific r, respectively c, comprises the complete input data to the network.

To obtain $\mathbf{z}$ of a vertex, part of the input (output) vector is allocated to represent the vertex label and the existence of pointers (edges) p to connected vertices. There are q pointer slots reserved, where $q = max\{degree(v)\}$, with $v \in V$. The input vector is composed of $t_h + q \cdot m \to \mathbb{R}^n$ elements, where t_h is the number of elements used to represent vertex information (label plus pointer existence), and m is the number of elements used to represent pointer values. If a vertex has less than q pointers, a nil pointer is used (Ellingsen 97). The hidden representation $\mathbf{z}$ of a specific vertex is understood as the pointer for that vertex. As part of other input vectors, it will thus be used as pointer and iteratively fed to the network. Eventually, we obtain the collection $\mathbf{Z}$ of fixed-sized vectors which represents all ontological entities.

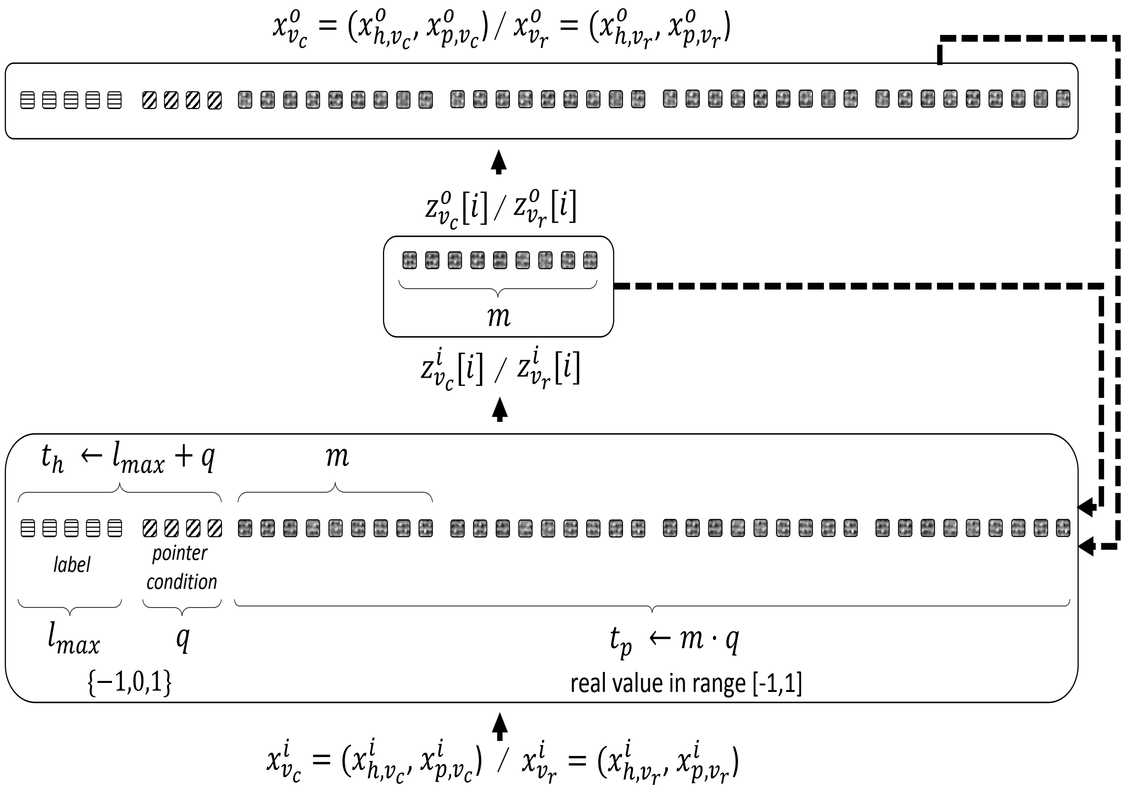

$x^o_{v_c} = (x^o_{h,v_c}, x^o_{p,v_c}) / x^o_{v_r} = (x^o_{h,v_r}, x^o_{p,v_r})$
$z^o_{v_c}[i] / z^o_{v_r}[i]$
m
$z^i_{v_c}[i] / z^i_{v_r}[i]$
$t_h \leftarrow l_{max} + q$
m
label
pointer condition
l_{max}
q
$\{-1,0,1\}$
$t_p \leftarrow m \cdot q$
real value in range [-1,1]
$x^i_{v_c} = (x^i_{h,v_c}, x^i_{p,v_c}) / x^i_{v_r} = (x^i_{h,v_r}, x^i_{p,v_r})$

Figure 6.2: Experimental LRAAM implementation.

For each reference ontology r, we can thus determine the most similar compound ontology c by choosing $\bar{c} \in C$, such that $d(\mathbf{Z}^{\hat{g}}_{r}, \mathbf{Z}^{\hat{g}}_{\bar{c}}), \forall r \in R, \forall c \in C : \mathbf{Z}^{\hat{g}}_{r} \times \mathbf{Z}^{\hat{g}}_{c} \to \mathbb{R}^m$ is minimized.

In a final step, $\bar{c}$ is selected for effective integration. Again, the assumption is that the selected service sequence $\overline{ss'}$ is semantically closer to the sequence of activities aa' and thus better suited to support it. By extension, data attributes of s and s' should more easily map each other, e.g., by symbolic matching techniques, such as described in Section 5.3.

The next chapter specifies the algorithm as it is implemented to achieve the collaboration of the symbolic and distributed approaches described before.

7

The LRAAM-based matching algorithm

In order to test the above described scenario, a tool – *OntoProc* – was developed. *OntoProc* runs GSSC as described in Chapter 4. It also executes the LRAAM matching method (cf., place holder algorithm 4.2.4). We call the corresponding algorithm LDIST. LDIST receives, transforms, and processes r and c. Specifically, it computes distance values for d_{rc}. To this end, *OntoProc* instruments the open source library Neuroph[1]. The latter realizes the LRAAM topology.

7.1 Definition of parameters and variables

In general, *OntoProc* receives the following input data:

- A process ontology and two or more service ontologies in order for *OntoProc* to construct $r \in R$ and $c \in C$ (GSSC).
- LRAAM specific input parameters (LDIST, inferred from (Ellingsen 97)):
  - ϵ: total error to reach before training stops.
  - η: backpropagation learning rate.
  - σ_z: slope of hidden layer activation function $\varphi(\upsilon) = 1/(1 + \mathrm{e}^{\sigma_z \upsilon})$, with υ the sum of input to the unit.

[1]http://neuroph.sourceforge.net/

- σ_h: slope of the activation function $\varphi(v) = 1/(1 + \mathrm{e}^{\sigma_h v})$ of the label part of input/output vector $\mathbf{x}$, being $\mathbf{x}_h$.
- σ_p: slope of the activation function $\varphi(v) = 1/(1 + \mathrm{e}^{\sigma_p v})$ of the pointer part of input/output vector $\mathbf{x}$, being $\mathbf{x}_p$.
- m: freely defined number of hidden layer nodes, with $m < |x|$. Note: It determines the size of the input vector x (cf., Fig. 6.2).

Based on this input data, *OntProc* initializes the following variables in order to execute LDIST presented thereafter:

- $V_r = \{v_r\}$, the set of vertices of the reference ontology, where v_r is a vertex in the graph representation of reference ontology $r \in R$.
- $V_c = \{v_c\}$, the set of vertices of the compound ontology, where v_c is a vertex of the graph representation of compound ontology $c \in C$.
- $E_{v_r} = \{v_r'\}$, such that an edge starts from v_r and ends at v_r'.
- $E_{v_c} = \{v_c'\}$, such that an edge starts from v_c and ends at v_c'.
- $q \leftarrow \mathit{maxOutDegree}(V_r \cup V_c)$, the maximum number of connection slots to include in the input and output vectors of the LRAAM.
- $l_{max} \leftarrow \mathit{maxLabelLength}(V_r \cup V_c)$, the maximum number of $\{-1, 0, 1\}$ values required to encode the labels of the vertices.
- $t_h = q + l_{max}$, the number of $\{-1, 0, 1\}$ values required in the input and output vectors to encode the different vertices.
- $t_p \leftarrow m \cdot q$, the number of real values required in the input and output vectors to encode the connections between vertices (pointers).
- $\mathbf{z}^i_{v_r}$ is the reduced representation of vertex $v_r \in V_r$ at the beginning of a training iteration. It is a vector of m values.
- $\mathbf{z}^o_{v_r}$ is the reduced representation of vertex $v_r \in V_r$ at the end of a training iteration. It is a vector of m values.

- $\mathbf{x}^i_{v_r} = (\mathbf{x}^i_{h,v_r}, \mathbf{x}^i_{p,v_r})$ is the input vector of vertex $v_r \in V_r$ at the beginning of a training iteration, where $\mathbf{x}^i_{h,v_r}$ is a $\{-1, 0, 1\}$ value vector encoding the label and pointer conditions of vertex v_r and $\mathbf{x}^i_{p,v_r}$ is a real vector encoding the pointers outgoing from vertex v_r.

- $\mathbf{x}^o_{v_r} = (\mathbf{x}^o_{h,v_r}, \mathbf{x}^o_{p,v_r})$ is the output vector of vertex $v_r \in V_r$ at the beginning of a training iteration, where $\mathbf{x}^o_{h,v_r}$ is a $\{-1, 0, 1\}$ value vector encoding the label and pointer conditions of vertex v_r and $\mathbf{x}^o_{p,v_r}$ is a real vector encoding the pointers outgoing from vertex v_r.

- $\mathbf{z}^i_{v_c}$ is the reduced representation of vertex $v_c \in V_c$ at the beginning of a training iteration. It is a vector of m values.

- $\mathbf{z}^o_{v_c}$ is the reduced representation of vertex $v_c \in V_c$ at the end of a training iteration. It is a vector of m values.

- $\mathbf{x}^i_{v_c} = (\mathbf{x}^i_{h,v_c}, \mathbf{x}^i_{p,v_c})$ is the input vector of vertex $v_c \in V_c$ at the beginning of a training iteration, where $\mathbf{x}^i_{h,v_c}$ is a $\{-1, 0, 1\}$ value vector encoding the label and pointer conditions of vertex v_c and $\mathbf{x}^i_{p,v_c}$ is a real vector encoding the pointers outgoing from vertex v_c.

- $\mathbf{x}^o_{v_c} = (\mathbf{x}^o_{h,v_c}, \mathbf{x}^o_{p,v_c})$ is the output vector of vertex $v_c \in V_c$ at the beginning of a training iteration, where $\mathbf{x}^o_{h,v_c}$ is a $\{-1, 0, 1\}$ value vector encoding the label and pointer conditions of vertex v_c and $\mathbf{x}^o_{p,v_c}$ is a real vector encoding the pointers outgoing from vertex v_c.

7.2 The matching algorithm

LDIST is defined below. Its final result are values for the distance measure d_{rc} for each pair $r \times c$. Again, these values are used by GSSC (cf., 4.2.1) for subsequent global optimization. For readability, we divided LDIST into several parts:

7.2.1 Input/output specification

Algorithm 7.2.1: LDIST - *general specifications*

Goal : Determine semantic (ontological) distance of each pair of reference and compound ontologies by means of a "Labelled Recursive Auto-Associative Memory."

Input :

- Ontology-specific: $V_r = \{v_r\}$, $V_c = \{v_c\}$, $E_{v_r} = \{v'_r\}$ $E_{v_c} = \{v'_c\}$,
- LRAAM-specific: ϵ, η, σ_z, σ_h, σ_p, m:

Output : d_{rc}

Logic :

```
1 LDIST - initialize $v_r$                                           // Algo. 7.2.2
2 LDIST - initialize $v_c$                                           // Algo. 7.2.3
3 LDIST - create LRAAM instance for $r \in R$, initialize first input vector   // Algo. 7.2.4
4 LDIST - create LRAAM instance for $r \in R$, initialize first output vector  // Algo. 7.2.5
5 LDIST - train LRAAM for $r$                                        // Algo. 7.2.6
6 LDIST - create LRAAM instance for $c \in C$, initialize first input vector   // Algo. 7.2.7
7 LDIST - create LRAAM instance for $c \in C$, initialize first output vector  // Algo. 7.2.8
8 LDIST - train LRAAM for $c$                                        // Algo. 7.2.9
9 LDIST - compute distance $d_{rc}$                                  // Algo. 7.2.10
```

7.2.2 Vertex initialization v_r

Algorithm 7.2.2: LDIST - *initialize* v_r

We initialize the first reduced representation of each vertex of the reference ontology with random values.

```
foreach $v_r \in V_r$ do
    for $i$ from $0$ to $m$ (excl.) do
        $\mathbf{z}^o_{v_r}[i] \leftarrow$ a real random value in range $[-1, 1]$.
```

7.2.3 Vertex initialization v_c

Algorithm 7.2.3: LDIST - *initialize* v_c

We initialize the first reduced representation of each vertex of the compound ontology. For those vertices which correspond to an element of the sequence ontology, we use $\mathbf{z}^o_{v_r}$ from the corresponding vertex v_r in the reference ontology. Otherwise, we use random values.

```
foreach $v_c \in V_c$ do
    if $v_c$ is a vertex from the sequence ontology then then
        $v_r \leftarrow$ the corresponding vertex in the reference ontology.
        $\mathbf{z}^o_{v_c} \leftarrow \mathbf{z}^o_{v_r}$.
    else
        for $i$ from $0$ to $m$ (excl.) do
            $\mathbf{z}^o_{v_c}[i] \leftarrow$ a real random value in range $[-1, 1]$.
```

7.2.4 Create LRAAM instance for $r \in R$ (initialize first input vector)

Algorithm 7.2.4: LDIST - *create LRAAM instance for* $r \in R$, *initialize first input vector*

We initialize the input vector in order to bootstrap the training process. We only need to initialize the label value to its binary representation, replacing 0 by -1, and using 0 as padding, and the pointer conditions to 0 or 1.

```
1 foreach $v_r \in V_r$ do
2     $\mathbf{x}^i_{h,v_r}[0, l_{max} - 1] \leftarrow$ binary representation of the label of vertex $v_r$, replacing each
      0 by $-1$, then padding with 0.
3     for $i$ from 0 to $||E_{v_r}||$ (excl.) do
4         $\mathbf{x}^i_{h,v_r}[l_{max} + i] \leftarrow 1$
5     for $i$ from $||E_{v_r}||$ to $n$ (excl.) do
6         $\mathbf{x}^i_{h,v_r}[l_{max} + i] \leftarrow 0$
```

7.2.5 Create LRAAM instance for $r \in R$ (initialize first output vector)

Algorithm 7.2.5: LDIST - create LRAAM instance for $r \in R$, *initialize first output vector*

We initialize the first output vector in order to bootstrap the training process. We only need to initialize the *nil* vectors to random values for the output vector.

```
1 foreach $v_r \in V_r$ do
2     for $i$ from $||E_{v_r}||$ to $n$ (excl.) do
3         for $j$ from 0 to $m$ (excl.) do
4             $\mathbf{x}^o_{p,v_r}[t_h + i \cdot m + j] \leftarrow$ a real random value in the $[-1, 1]$ range.
```

7.2.6 LRAAM training for $r \in R$

Algorithm 7.2.6: LDIST - *train LRAAM for r*

We start the training process of the LRAAM at this point. We perform the training loop until the ending conditions are reached, that is, the error is below the expected threshold.

```
1  $\Delta \leftarrow \infty$
2  while $\Delta \geq \epsilon$ do
       We initialize the input vector of the current iteration using the output vector and the
       reduced representations from the previous iteration.
3      foreach $v_r \in V_r$ do
           We replace the pointer of each vertex with the last calculated reduced
           representation of that vertex.
4          $i \leftarrow 0$
5          foreach $v'_r$ in $E_{v_r}$ do
6              $\mathbf{x}^i_{p,v_r}[i \cdot m, ((i+1) \cdot m) - 1] \leftarrow \mathbf{z}^o_{v'_r}$        // replacing the pointer
7              $i \leftarrow i + 1$
8          We copy all nil pointers from the previous output vector into the current
           input vector.
9          $\mathbf{x}^i_{p,v_r}[i \cdot m, (n \cdot m) - 1] \leftarrow \mathbf{x}^o_{p,v_r}[i \cdot m, (n \cdot m) - 1]$  // copying nil pointers
10     $\mathbf{z}^i_{v_r} \leftarrow \mathbf{z}^o_{v_r}$
       We train the LRAAM.
11     The LRAAM is trained by backpropagation with input vector set $\{\mathbf{x}^i_{v_r}\}$.
       After training, we update the output vectors using the newly trained LRAAM.
12     $\Delta \leftarrow 0$
13     foreach $v_r \in V_r$ do
14         $\mathbf{x}^o_{v_r} \leftarrow outputLayer(LRAAM(\mathbf{x}^i_{v_r}))$
15         $\mathbf{z}^o_{v_r} \leftarrow hiddenLayer(LRAAM(\mathbf{x}^i_{v_r}))$
16         $\Delta \leftarrow \Delta + euclidianDistance(\mathbf{z}^i_{v_r}, \mathbf{z}^o_{v_r})$
```

7.2.7 Create LRAAM instance for $c \in C$ (initialize first input vector)

Algorithm 7.2.7: LDIST - *create LRAAM instance for c* $\in C$, *initialize first input vector*

We initialize the input vector in order to bootstrap the training process. We only need to initialize the label value and the pointer conditions to 0 or 1.

```
1 foreach $v_c \in V_c$ do
2     $\mathbf{x}^i_{h,v_c}[0, l_{max} - 1] \leftarrow$ binary representation of the label of vertex $v_c$, replacing each
      0 by $-1$, then padding with 0.
3     for i from 0 to $||E_{v_c}||$ (excl.) do
4         $\mathbf{x}^i_{h,v_c}[l_{max} + i] \leftarrow 1$
5     for i from $||E_{v_c}||$ to n (excl.) do
6         $\mathbf{x}^i_{h,v_c}[l_{max} + i] \leftarrow 0$
```

7.2.8 Create LRAAM instance for $c \in C$ (initialize first output vector)

Algorithm 7.2.8: LDIST - *create LRAAM instance for c* $\in C$, *initialize first output vector*

We initialize the first output vector in order to bootstrap the training process. We only need to initialize the *nil* vectors to random values for the output vector.

```
1 foreach $v_c \in V_c$ do
2     for i from $||E_{v_c}||$ to n (excl.) do
3         for j from 0 to m (excl.) do
4             $\mathbf{x}^o_{p,v_c}[t_h + i \cdot m + j] \leftarrow$ a random value in the $[-1, 1]$ range.
```

7.2.9 LRAAM training for $c \in C$

Algorithm 7.2.9: LDIST - *train LRAAM for c*

We start the training process of the LRAAM at this point. We perform the training loop until the ending conditions are reached, that is, the error is below the expected threshold.

```
1  $\Delta \leftarrow \infty$
2  while $\Delta \geq \epsilon$ do
       We initialize the input vector of the current iteration using the output vector and the
       reduced representations from the previous iteration.
3      foreach $v_c \in V_c$ do
           First we replace the pointer of each vertex with the last calculated reduced
           representation of that vertex.
4          $i \leftarrow 0$
5          foreach $v'_c$ in $E_{v_c}$ do
6              $\mathbf{x}^i_{p,v_c}[i \cdot m, ((i+1) \cdot m) - 1] \leftarrow \mathbf{z}^o_{v'_c}$.
7              $i \leftarrow i + 1$.

           We copy all nil pointers from the previous output vector into the current
           input vector.

8          $\mathbf{x}^i_{p,v_c}[i \cdot m, (n \cdot m) - 1] \leftarrow \mathbf{x}^o_{p,v_c}[i \cdot m, (n \cdot m) - 1]$.
9      $\mathbf{z}^i_{v_c} \leftarrow \mathbf{z}^o_{v_c}$

       We train the LRAAM

10     The LRAAM is trained by backpropagation with input vector set $\{\mathbf{x}^i_{v_c}\}$.

       After training, we update the output vectors using the newly trained LRAAM.

11     $\Delta \leftarrow 0$
12     foreach $v_c \in V_c$ do
13         $\mathbf{x}^o_{v_c} \leftarrow outputLayer(LRAAM(\mathbf{x}^i_{v_c}))$
14         $\mathbf{z}^o_{v_c} \leftarrow hiddenLayer(LRAAM(\mathbf{x}^i_{v_c}))$
15         $\Delta \leftarrow \Delta + euclidianDistance(\mathbf{z}^i_{v_c}, \mathbf{z}^o_{v_c})$
```

7.2.10 Calculate distance d_{rc}

Algorithm 7.2.10: LDIST - *compute distance* d_{rc}

We calculate the Euclidean distance.

```
1 $d_{rc} \leftarrow 0$
2 foreach $v_c \in V_c$ do
3     if $v_c$ is a vertex from the sequence ontology then
4         $v_r \leftarrow$ the corresponding vertex in the reference ontology.
5         $d_{rc} \leftarrow euclidianDistance(\mathbf{z}^o_{v_r}, \mathbf{z}^o_{v_c})$
6 Return $d_{rc}$
```

7.3 Relation to the global algorithm GSSC

After calculation of each individual distance d_{rc}, GSSC (cf., Algo. 4.2.4, *computeDistance-Matrix*) constructs the distance matrix D, and thereafter the optimal set of services to support all related business requirements (i.e., the business process). Alternatively, we could calculate only the distances required as the branch and bound process of GSSC proceeds.

In the next chapter, we describe several experiments to investigate the above proposed matching method.

8

Experimental methodology and results

The following sections describe our methodological approach to test the use of LRAAM for calculating the distance measure d_{rc}. It is followed by a presentation of the experiments' results.

8.1 Experimentation setup

We conducted two experiments. The first experiment is divided into a preliminary and three main phases. With *OntoProc*, we seek LRAAM parameters which maximize classification performance. The second experiment is separated into two sub-experiments. Those two experiments apply parameter combinations found in the first experiment to more complex input data. The two sub-experiments are aligned to the running example.

8.1.1 Introductory experiment

This experiment uses simple input data to investigate the LRAAM's classification performance.

Preliminary phase

In the preliminary phase, the input data is prepared. It consists of a set of xml-based files, namely a set of process and service ontologies. The ontological standard is OWL. We use the Protégé editor[1] to create these files.

[1]http://protege.stanford.edu/

OntoProc receives a process ontology with two activities and their order. Each activity is described by exactly one label. The labels are $CreditChecking$ and $CreditApproving$ (cf., Fig. 8.1).

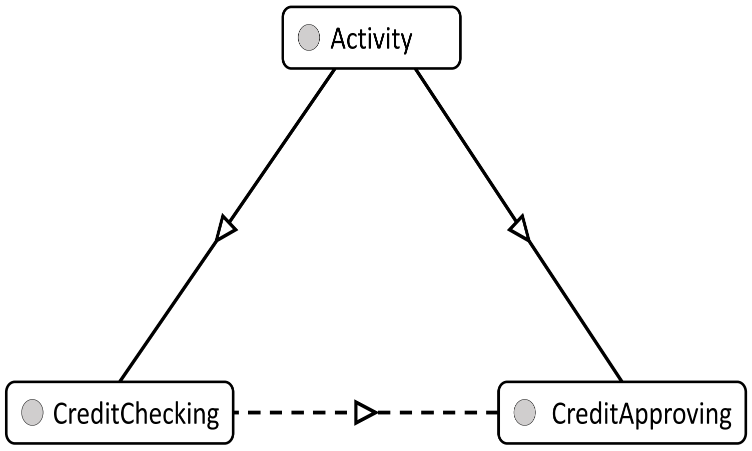


Figure 8.1: Experimental process ontology

Furthermore, two simple service ontologies are prepared. Each service is described by exactly one label. The labels are also $CreditChecking$ and $CreditApproving$ (cf., Fig. 8.2).

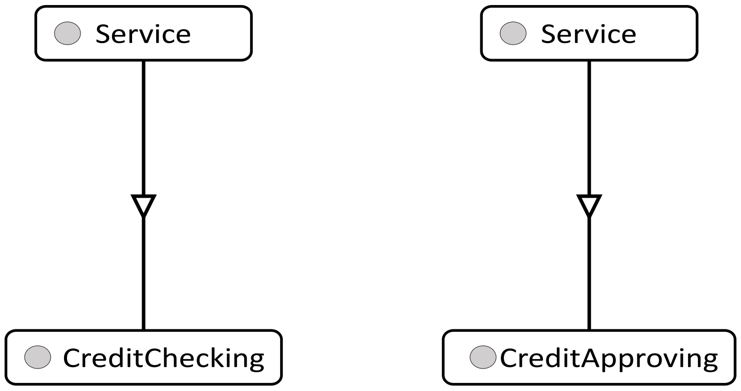


Figure 8.2: Experimental service ontologies

The three ontologies are loaded into *OntProc*, which generates three new ontologies, namely the reference ontology $r : CreditChecking$ x $CreditApproving$, and two compound ontologies $c_1 : CreditApproving$ x $CreditChecking$ & $c_2 : CreditChecking$ x $CreditApproving$. Note that at this point the LRAAM's classification performance is explored. The experiment is thus not extended to more than two related activities, hence multiple reference ontologies.

In a further step, the ontologies r and c_i are used to implement the LRAAM to calculate d_{rc_i}, with $i = \{1, 2\}$ for further analysis. For this, the following variables are defined:

- w: the expected winning service, with $w \in \{1, 2\}$. For the controlled experiment, we expect $w = 2$, as $r \equiv c_2$.
- d_i: distance of service combination c_i, with $i = \{1, 2\}$, and r.
- d_w: distance of expected winning service combination c_w and r
- $\mathfrak{r}_i$: ranking of service combination i out of all combinations.
- $\mathfrak{r}_w$: ranking of expected winning service combination out of all combinations.
- b: 1 if correctly matched, i.e., $w = 2$, 0 otherwise.

Phase 1

In Phase 1, the relationship between the LRAAM input parameters $\epsilon, \eta, \sigma_z, \sigma_h, \sigma_p, m$ and output d_i is analyzed. For each parameter, interval and tick size is fixed to explore reasonable, yet limited, parameter combinations within LRAAM's large state space. Tick size is the value of each increment of the parameter value from lowest to largest value in the interval. For example, an interval from 0 to 10, with tick size of 5 would test values in the set $\{0, 5, 10\}$.

In this phase, the following parameter intervals and tick sizes are fixed:

- Epsilon (ϵ): interval [0.15,0.25], increment 0.05
- Learning rate (η): interval [0.1,0.3], increment: 0.1
- Hidden Layer Slope (σ_z): interval [0.5,1.5], increment: 0.5
- I/O Layer Binary Slope (σ_h): interval [5,7], increment: 1
- I/O Layer Real Slope (σ_p): interval [0.5,1.5], increment: 0.1
- Number Hidden Layer Nodes (m): interval [15,75], increment: 5

All parameter values are recombined with each other. To control processing time, one LRAAM instance per combination is executed. It yields d_i as c_i is compared to r. A list of d_i for all combinations is the result. On it, a multiple linear regression is conducted in order to identify the parameters having a significant correlation with d_i. These parameters will be analyzed further in subsequent phases. The other parameters are then fixed heuristically at a value which maximizes b for d_w (i.e., the number of good classifications). At this point, classification performance is not yet evaluated.

Phase 2

In Phase 2, we further explore the impact of parameters with a significant correlation with d_i. Specifically, the interval is increased and the tick size is decreased. More of LRAAM's state space is thus explored, this time to find optimal parameter values. All resulting parameter values are recombined with each other. Similar to Phase 1, one LRAAM instance per combination is executed to yield d_i. A list of d_i for all combinations is the result. Based on this list, each variable parameter is heuristically analysed for promising values, i.e., which maximize b for d_w. Classification performance is still not evaluated.

Phase 3

In Phase 3, 2000 LRAAM runs are executed for these promising values found in Phase 2. Each combination results in a list of $2000 \cdot d_i$, which is analysed for classification performance. Specifically, each list is transformed by assigning a ranking $\mathfrak{r}_i$ to d_i. The smallest of two d_i per run is given a ranking $\mathfrak{r}_i = 1$. The other ranking ($\mathfrak{r}_i = 2$) is, furthermore, transformed into a weighted ranking, normalized over the maximum distance spread over all runs. It gives a more fine-tuned account of the classification performance.

The general success criteria is defined as, $\mu_{\mathfrak{r}_w} < \mu_{\mathfrak{r}_i}, \forall i \neq w$ which in the experiment is $\mu_{\mathfrak{r}_2} < \mu_{\mathfrak{r}_1}$. In other words, the mean ranking of the expected winning service combination c_w must be smaller than the respective means of all other service combinations c_i.

If a list respects the criterium, it is selected and tested for statistical significance. To this end, an independent samples t-test is used, as the sample groups i are independent of each other. If the null-hypothesis ($\alpha = 0.05$) can be rejected, d_w is correctly classified. In that case, the best combination of services (c) to support related business activities (r) could be identified.

8.1.2 Running example experiments

For both following sup-experiments (e_1 and e_2), we add an additional service ontology. For convenience, we name the existing services s_1, s_2, and the new service s_3. It stands for the more sophisticated credit checking service. In the first sub-experiment, we feed the process ontology (p_1) from the introductory experiment. In the second sub-experiment, we use a different process ontology (p_2). It stands for a different business requirement.

Preliminary phase

e_1 : *OntoProc* uses p_1 and s_1, s_2, and s_3. s_3 contains an additional concept $Awesome$ $BigDataRiskCalculation$ as subclass (cf., Fig. 8.3). It shall simulate a more sophisticated scoring service offering.

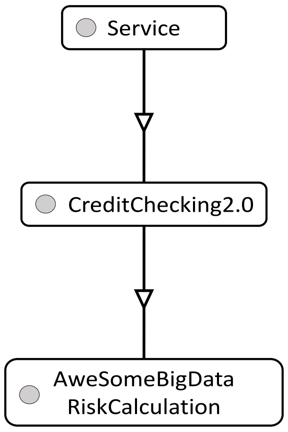


Figure 8.3: Alternative service for selection

The four ontologies are loaded into *OntoProc*. From those, it generates seven new ontologies, namely a reference ontology r : $CreditChecking$ x $CreditApproving$, and six compound ontologies c_1 : $CreditApproving$ x $CreditChecking$, c_2 : $CreditApproving$ x $CreditCreditChecking2.0$, c_3 : $CreditChecking$ x $CreditApproving$, c_4 : $CreditChecking$ x $CreditChecking2.0$, c_5 : $CreditChecking2.0$ x $CreditApproving$, c_6 : $CreditChecking2.0$ x $CreditChecking$. Note that the set C of compound ontologies consists of all possible service combinations.

e_2 : *OntoProc* uses s_1, s_2, and s_3 and p_2 (cf., Fig. 8.4). For the latter, the label $CreditChecking$ is changed to $CreditChecking2.0$. Furthermore, a subclass $CoolBigDataDefaultScore$ is added. It is aligned to our running example, expressing a new business requirement. The activity $CreditApproving$ remains stable.

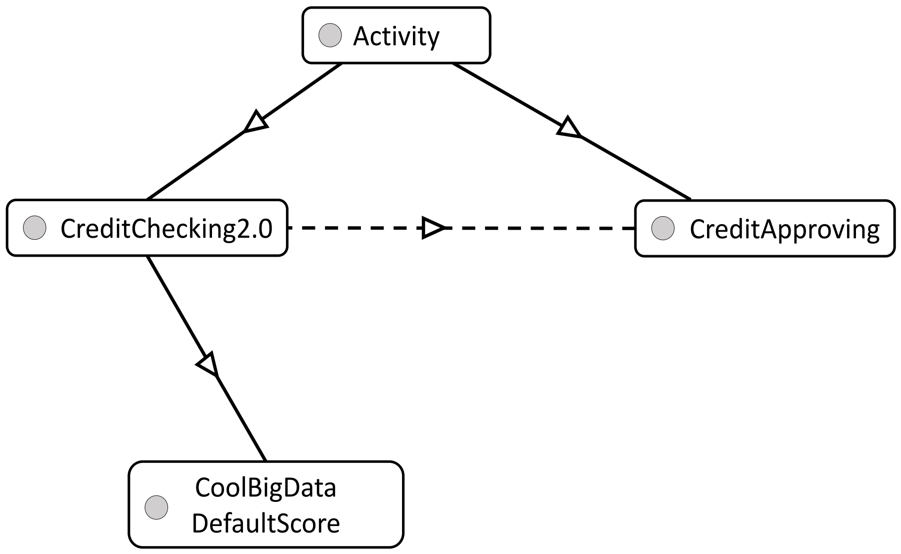


Figure 8.4: Experimental process ontology (enhanced)

Similar to e_1, four ontologies are loaded. *OntoProc* generates seven new ontologies, namely a reference ontology r : $CreditChecking2.0$ x $CreditApproving$, and six compound ontologies c_1 : $CreditApproving$ x $CreditChecking$, c_2 : $CreditApproving$ x $CreditCreditChecking2.0$, c_3 : $CreditChecking$ x $CreditApproving$, c_4 : $CreditChecking$ x $CreditChecking2.0$, c_5 : $CreditChecking2.0$ x $CreditApproving$, c_6 : $CreditChecking2.0$ x $CreditChecking$.

For both, e_1 and e_2, the ontologies r and c_i are used to implement the LRAAM to calculate d_{rc_i}, with $i = \{1, 2, 3, 4, 5, 6\}$ The same variables as mentioned in the first experiment are defined:

- w: the expected winning service, with $w \in \{1, 2, 3, 4, 5, 6\}$. For the controlled experiment. For e_1, we expect $w = 5$, as $r \approx c_5$. For e_2, we expect $w = 3$, as $r \approx c_4$.
- d_i: distance of service combination c_i, with $i = \{1, 2, 3, 4, 5, 6\}$, and r.
- d_w: distance of expected winning service combination c_w and r
- $\mathfrak{r}_i$: ranking of service combination i out of all combinations.
- $\mathfrak{r}_w$: ranking of expected winning service combination out of all combinations.

- b: 1 if correctly matched, i.e., $w = 5$ (e_1) and $w = 4$ (e_2), 0 otherwise.

Verification phase

For e_1 as for e_2, similar to Phase 3 of the introductory experiment, 2000 LRAAM runs are executed for promising values found in Phase 2 of the introductory experiment. Each combination results in a list of $2000 \cdot d_i$, which is analysed for classification performance. Specifically, each list is transformed by assigning a ranking $\mathfrak{r}_i$ to d_i. The smallest of six d_i per run is given a ranking $\mathfrak{r}_i = 1$. Depending on the value of d_i, the other rankings consequently correspond to 2 to 6. We do not assign further weighting to the rankings.

The general success criteria is defined as, $\mu_{\mathfrak{r}_w} < \mu_{\mathfrak{r}_i}, \forall i \neq w$ which in both experiments is $\mu_{\mathfrak{r}_5} < \mu_{\mathfrak{r}_i}$. In other words, the mean ranking of the expected winning service combination c_w must be smaller than all respective means of all other service combinations c_i.

If this criterium is respected, all lists in e_1, respectively e_2, must still be tested for statistical significance. Again, an independent samples t-test is used, as the sample groups i are independent of each other. If the null-hypothesis ($\alpha = 0.05$) for each $\mu_{\mathfrak{r}_i}$ can be rejected, d_w is correctly classified. In that case, the best combination of services (c) to support related business activities (r) could be identified.

8.2 Experimental results

The following sections describe the results for the introductory experiment, as well as for e_1 and e_2. We begin with the introductory experiment.

8.2.1 Introductory experiment

Phase 1

In Phase 1, a list of 12,636 d_i is generated, corresponding to 6,318 parameters value combinations. The model summary of the conducted multiple regression is shown in Table 8.1:

The predictive capacity of η (learning rate) and m (number of hidden layer nodes) is high as these parameters explain a large part of d's variability ($R^2 = 83.8\%$). Both parameters are promoted to Phase 2. As the other parameters (IO layer binary slope σ_h, IO layer real slope σ_p) do not have a significant impact on d, they are fixed based on suggestions made

Table 8.1: Multiple regression model summary.

Predictors	**R**	**R^2**	**Adj. R^2**	**Std.Err.of Est.**
η	0.857	0.735	0.735	0.58527
η & m	0.915	0.838	0.838	0.45722
η & m & σ_h	0.949	0.900	0.900	0.35947
η & m & σ_h & σ_p	0.949	0.900	0.900	0.35935

in (Ellingsen 97). We have $\epsilon = 0.15$ (epsilon), $\sigma_h = 6$ (IO layer binary slope), $\sigma_p = 0.5$ (IO layer real slope), and $\sigma_z = 0.5$ (hidden layer slope).

For σ_z, σ_h, and σ_p, the values selected also yield the highest count of correctly mapped c in the list (cf., marked area in Table 8.2).

Table 8.2: Parameter combination with highest count of correctly mapped c (framed).

Epsilon	**IO layer binary slope**	**IO layer real slope**	**Hidden layer slope**	**Correct matches**
0.15	6	0.50	0.50	29
0.20	6	1.50	0.40	26
0.20	7	1.00	0.70	26
0.15	5	1.50	0.70	25
0.25	5	1.00	0.30	25
0.25	7	0.50	0.70	25

Phase 2

In Phase 2, the interval of m and η is increased to cover more of LRAAMs state space. A list of 6,060 d_i is generated, corresponding to 3,030 parameter value combinations. From this list, zones of promising matching performance are identified (cf., Fig. 8.5 and 8.6).

The values $\eta = \{0.09, 0.20\}$ and $m = \{22, 42, 105\}$ are fixed for further processing.

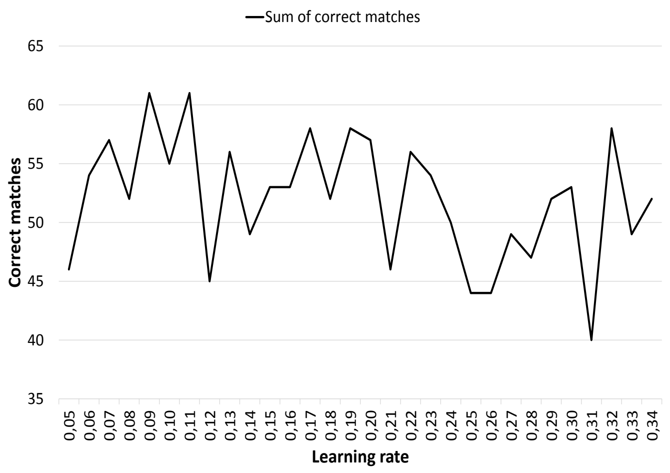


Figure 8.5: Phase 2: Correct matches for η's tested interval.

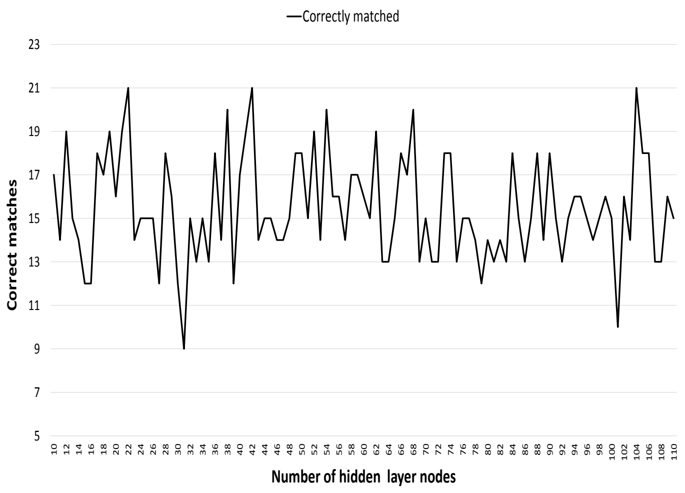


Figure 8.6: Phase 2: Correct matches for m's tested interval.

Phase 3

Based on the above values, six combination trials of 2,000 runs are executed during Phase 3. The results are shown in Table 8.3.

Table 8.3: Classification results for fixed parameter value combinations.

Parameters		Matches	
η	m	c_1	c_2 (c_w)
0.09	22	989	**1,011**
0.20	22	949	**1,051**
0.09	42	996	**1,004**
0.20	42	1,005	995
0.09	105	974	**1,026**
0.20	105	1,042	958

Except for the combinations $(0.2, 42)$ and $(0.2, 105)$, the classifications passed the success criterium, namely, $\mu_{\mathfrak{r}_w} < \mu_{\mathfrak{r}_i}$. The significance of the results is verified by an independent samples t-test (cf., Table 8.4).

Table 8.4: t-test for significance of $\mu_{\mathfrak{r}_w} < \mu_{\mathfrak{r}_i}$

Parameters		Levene's test		t-test for equality of μ		
η	m	F	Sig.	t	df	Sig.(2-tailed)
0.09	22	0.792	0.374	1.122	3,998	0.262
0.09	42	0.368	0.544	0.587	3,998	0.557
0.09	105	0.000	0.991	1.148	3,998	0.251
0.20	22	2.232	0.135	3.025	3,998	0.002*

* H_0 rejected.

The first three parameter combinations shown in Table 8.4 do not indicate a significant classification success. However, for the parameter combination (0.20, 22), a very low p-value of 0.002 could be identified. It suggests that the mean ranking of compound ontology $c_2 = 1.89$ is significantly different from the mean ranking of $c_1 = 2.02$. With $\alpha = 0.05$, in this case, the null-hypothesis is thus rejected. In other words, with 2,000 runs, the combination correctly classifies the input. However, SPSS only allows for 2-tailed t-tests, whereas we are interested only in the left tail of the t-distribution. Since the t-test supposes a symmetrical distribution, in this case, the "significant" tail will reflect significance at $p/2 = 0.025$ or below[1]. As $0.002 < 0.025$, the validity of rejection holds.

8.2.2 Running example experiments

For obvious reasons, namely the significance result, we ought to use the parameter combination (0.20, 22). However, we also test three other promising parameter combinations, namely (0.09, 22), (0.09, 42), and (0.20, 42). It is due to the non-deterministic nature of the LRAAM. We seek a better understanding of *OntoProc*'s behavior.

Verification phase e_1

Based on the above parameter combination, four combination trials of 2,000 runs are executed. The results are shown in Table 8.5.

Table 8.5: Classification results for fixed parameter value combinations.

Parameters		Matches					
η	m	c_1	c_2	c_3 (c_w)	c_4	c_5	c_6
0.09	22	591	223	589	197	195	205
0.20	22	602	194	**627**	155	211	211
0.09	42	833	83	**860**	70	58	49
0.20	42	889	55	**903**	46	82	63

Except for the combination $(0.09, 22)$, the classifications passed the success criterium, namely, $\mu_{\mathfrak{r}_w} < \mu_{\mathfrak{r}_i}$, with $i = \{1, 2, 4, 5, 6\}$ (marked bold). The significance of the results is verified by an independent samples t-test. In Table 8.6, for each parameter combination, and for each combination r and c_i per parameter combination, significance tests are conducted and shown[2].

[1]http://www-01.ibm.com/support/docview.wss?uid=swg21476176 (visited May 19, 2015).
[2]For convenience, we do not show Levene's test for independence of variance anymore.

Table 8.6: t-test for significance of $\mu_{\mathfrak{r}_w} < \mu_{\mathfrak{r}_i}$, with $w = 3$

Parameters		**t-test for equality of μ_i**									
η	m	$\mu_{\mathfrak{r}_3} < \mu_{\mathfrak{r}_1}$		$\mu_{\mathfrak{r}_3} < \mu_{\mathfrak{r}_2}$		$\mu_{\mathfrak{r}_3} < \mu_{\mathfrak{r}_4}$		$\mu_{\mathfrak{r}_3} < \mu_{\mathfrak{r}_5}$		$\mu_{\mathfrak{r}_3} < \mu_{\mathfrak{r}_6}$	
		t	Sig.	t	Sig.	t	Sig.	t	Sig.	t	Sig.
0.20	22	0.608	**0.543**	28.168	0.000	32.099	0.000	28.728	0.000	29.133	0.000
0.09	42	0.791	**0.429**	51.778	0.000	58.073	0.000	51.252	0.000	58.782	0.000
0.20	42	0.380	**0.704**	59.629	0.000	67.047	0.000	61.025	0.000	65.690	0.000

For all parameter combination, the test rejects H_0, except for c_1 (marked bold). However, our success criterium dictates significance against all c_i. Therefore, despite a good classification for certain parameter combinations at first glance, overall significance could not be established.

Verification phase e_2

Based on the same parameter combination as used in e_1, four combination trials of 2,000 runs are executed. The results are shown in Table 8.7.

Table 8.7: Classification results for fixed parameter value combinations.

Parameters		**Matches**					
η	m	c_1	c_2	c_3	c_4	c_5 (c_w)	c_6
0.09	22	591	196	567	223	221	202
0.20	22	576	207	581	212	219	205
0.09	42	767	152	726	107	140	108
0.20	42	762	130	760	98	137	113

The expected winner, c_5, is not correctly classified. Significance tests are not necessary.

Overall, we can state that the experiments show inconclusive results. In the next chapter, we discuss major reasons for this. These reasons might serve as entry point for further research. To this end, we mention immediate changes to the experimental setup. We also suggest conceptual changes, which we deem promising in the longer run. Eventually, we give an answer to the second research question.

9

Discussion of the matching algorithm

With the implemented LRAAM oracle, we conducted experiments to verify similarity of service descriptions. During the experiments, we could identify some interesting results. For a simple use case (i.e., introductory experiment), significance under certain conditions was established. For more complex data input in experiment e_1 classification yielded apparent positive results. However, overall significance could not be verified. Experiment e_2 did not return positive classifications at all. However, for each parameter combination, we saw higher values of c_5 in e_2 than in e_1. It might indicate a tendency towards the right classification.

Hence, the results lead us to conclude that *OntoProc* needs further development and pruning.

In the next section, we depict several points where further immediate investigation is needed. Thereafter, we offer a research venue to address those points on a more general level. We conclude the chapter by restating and answering the second research question.

9.1 Architectural considerations

With respect to the LRAAM topology which is responsible for the mixed results, the following points caught our attention:

- Regarding the sigmoid activation function, we anticipated a higher impact of σ on d. In the multiple regression analysis, this could however not be shown. It warrants further analysis.

- The significant parameter combination $(0.2, 22)$ of the introductory experiment leads to a ratio $m/n = 22/151 = 0.15$. Based on (Ellingsen 97), a good classification performance was expected at a higher ratio of $m/n \approx 0.25$. It would thus suggest a higher number of hidden layer units for good results.

- For all experiments, each c and r contains same labels by default, namely those representing the sequence ontology, which we use to calculate the Euclidean distance. In the introductory experiments, only two specific labels are added to construct c, respectively r. In e_1 and e_2, one further label is used. These labels may not have sufficient impact to significantly alter LRAAM's rather volatile steady state. In other words, distinctiveness, which is sought, gets lost (cf., next point).

- We do not think that the current LRAAM implementation exhibits a sufficiently trustworthy steady state. Instead of initializing the input vector x with values from the interval $[-1, 1]$, similar to (Al-Said and Abdallah 09), it may be beneficial to use a smaller interval, e.g., $[-0.5, 0.5]$, to decrease potential variability, such as described in (Sperduti 93).

- Narrowing down LRAAM's state space is time-consuming. However, further intervals or parameters need to be explored, also combined with the above mentioned adaptations. One further parameter may be the number of consecutive times the LRAAM's establishes a steady state (expressed in $\mathbf{Z}$) based on the error tolerance ϵ, before it is chosen to calculate d. It would increase confidence in the validity of the steady state.

9.2 Adapting the LRAAM topology

The above list mainly points towards improvements of LRAAM's existing architecture, i.e., tuning the activation function, varying the ratio of the number of input and hidden layer nodes, etc. However, a further route to enhancing classification performance may reside in changing LRAAM's topology altogether. Specifically, the idea would be to couple the LRAAM with the *Deep Learning* (DL) approach (LeCun *et al.* 15).

The latter represents an ANN with up to 20 hidden layers (instead of only one). In total, such an ANN may contain tens or hundreds of thousands of units. Following (LeCun *et al.* 15), a DL system can implement extremely intricate functions of its inputs that are simultaneously sensitive to smallest variations. It exploits the property that many input signals are compositional hierarchies, in which higher-level features are obtained by composing lower-level ones.

The authors continue to state that similar hierarchies exist in speech and text from sounds to phones, phonemes, syllables, words and sentences. Clearly, this may prove beneficial to our research.

Finally, for DL, poor local minima are rarely a problem. Notwithstanding the initial conditions, i.e., the initialized random values, the system mostly reaches high quality solutions. To us, it could not only translate into a much more stable and trustworthy steady state $\mathbf{Z}$. It may also provide us with more comparable details on $\mathbf{z}$, as $\mathbf{z}$ itself would be decomposed into different hierarchies (corresponding to the number of hidden layers we would define to describe the compositional hierarchy of a pointer).

Bringing DL and LRAAM together thus seems a promising research venue.

9.3 LRAAM as alternative matching method

With the second research question we asked:

Other than classical symbol-based ontology matching, is an LRAAM, specifically its distributed patterns, a reliable alternative matching technique as part of an intelligent service selection and integration method?

Based on the experimentation, the current setup of the LRAAM does not allow for reliable service selection through vector matching. As mentioned above, the net's steady state is too volatile and susceptible to initial conditions, at least with respect to how we chose to initialize the input. Currently, we thus cannot positively answer the above question. However, the DL venue, briefly introduced above, would seemingly address the major points of criticism, namely volatility or sensitivity. Further investigations in this direction might yield better outcomes.

In relation to the this question, let's also remember that we could not answer the first research questions with yes, namely,

Can ontologies stand-alone, that is, in isolation, be used for an intelligent service selection and integration method?

It was due to the outlined restrictions of symbolic approaches as it concerns "intelligent" behavior of systems. However, if above issues were addressed by a combination of the LRAAM and the DL approach, this questions could also be looked at again. Hypothetically with such DL-LRAAM, unrestricted formal semantic descriptions of services would be set up. They would also merely be submitted to the language, such as OWL itself, but no further doctrine structure (such as WSMO, or SAWSDL), except for some key concepts. A DL-LRAAM could then reliably transform the complete representation (as investigated here) or perhaps only parts thereof for comparison. Based on (LeCun *et al.* 15), it could conduct a very sensitive matching of concepts, moving from the concept label to neighboring concepts, to embedding sub-graphs, to the complete representation, if necessary.

Thereby, it would only be relevant how detailed the descriptions are to “triangulate” their semantics, so to speak. In other words, where a single concept might be misinterpreted (e.g., a potential synonym, such as “Apple”), the same concept connected to another concept (e.g., “company”) becomes less ambiguous. With each additional concept related to the first concept, the risk of misinterpreting it as a fruit further decreases.

In general, we think such an approach would give the system more freedom for decision making, thereby almost simulating how humans would learn, recognize, and analyse (integration) features, should they change.

10

Conclusions

In this thesis, we introduced a service selection and composition (SSC) approach towards more intelligent, i.e., automatic yet flexible, integration of applications consumed as loosely coupled services. We discussed its necessity for organizations to cope with an increasing number of modularized, decentralized, and most importantly, *similar* services from which to choose. This services proliferation is due to the phenomenon of *long-term-zero profit equilibrium*, which we discussed and mentioned in Sections 1.1, 2.5, and 6.3.

We emphasized *standardization* and *flexibility*. On the one hand, standardization supports automation. Automation in turn helps decreasing human expert intervention. Beyond the direct cost and error factors, this would also mitigate the negative, again costly, effects of colliding experts' beliefs and perceptions (cf., Sect. 2.2). On the other hand, flexibility allows for *quick* on- and offboarding of services. It is needed to keep pace with ever more unforeseeable situations organizations face.

From Chapter 2, we learned that current syntactic integration techniques do not adequately respond to this challenge. They merely regulate information and focus on meta-data exchange. They strongly rely on human expert intervention. As other authors (e.g., (Born *et al.* 07, Fensel *et al.* 11)), we believe that ontologies, written in a language such as OWL, may better lend themselves as SSC building blocks. They stand for well-structured, formal-semantic knowledge representations. Those can be used by business experts to describe corporate business activities and by service providers to describe service offerings (cf., Chap. 3). Machines can then compare them through inferential matching (specifically cf., Sect. 3.3.3, 3.3.4, or for a wrap-up, Chap. 5). Clearly, it is a reasonable conjecture that an ontology describing a credit checking activity uses similar entities and relations as the needed credit checking service itself (provided that the same natural language is used to tag concepts in the ontology). Both possibly contain concepts, such as *Person* or *CreditRegister* including a relation *hasEntry*. In turn, an ontology describing a weather forecast service most probably does not include these concepts.

The symbolic fallacy: Ontology approaches are situated in the symbolic paradigm though. They situate on top of current industry standards such as WS and SOA. Instinctively, adoption and reusability seems a natural way to go. However, ontologies are still mere abstract symbol instances. Those do not convey an internal structure which is in any way relevant to its use (cf., Sect. 3.1). They are suitable when the property investigated is to be *either identical or non-identical* to other symbol instances[1]. They thus "...need to be bound to variables in *judiciously* chosen rules of inference (LeCun *et al.* 15)." Eventually, it limits execution, i.e., composition paths, and therefore flexibility, which again is assumed by human experts.

Therefore, we think that purely symbolic, either syntactic or formal-semantic, approaches will not provide us with the bridge between automation and flexibility. This is even more so in competitive environments. Here, changes in the input data of third-party services offerings can neither be anticipated, nor can technical control be exercised. Therefore to us, techniques such as introduced in Chapter 3, namely SAWSDL, OWL-S, or WSMO represent an impasse. They eventually lead into the Chinese room (cf., Sect. 3.1). Therein, automatic composition is very much possible. However, participating systems must be endowed with the same restrictive, syntactic meta structure (e.g., IOPEs, goals & capabilities).

Indeed, evidence for a lack of flexibility was given in Section 3.3.4: The mentioned techniques are not universally applicable. They presuppose specific, relatively stable, controlled environments, e.g., governmental ones (Kamaruddin *et al.* 12). To us, this fact still represents the very problem, the techniques claim to overcome, namely rigidity instead of flexibility.

The sub-symbolic venue: Our working assumption is that the chasm between automation and flexibility can be overcome by some sort of semantic grounding. For the context of this work, it means that the system develops a notion of the *signified*, rather than merely relying on arbitrary symbolic signifier for SSC. To this end, we investigated the non-deterministic artificial neural network (ANN) paradigm. We follow (LeCun *et al.* 15), who advocates to use activity vectors or weight matrices to enable "...a type of fast 'intuitive' inference that underlines effortless commonsense reasoning."

[1]Being the case virtually all the time.

Specifically, we looked at the LRAAM ANN-topology, presented in Chapters 6 and 7. Therein, knowledge representations, i.e., ontology graphs are encoded and stored in a distributed fashion over many weight vectors. We used the latter to compare the underlying ontological descriptions.

We emphasized that the comparison is no more conducted with arbitrary symbolic shapes. It is based on distributed, let's call it "grounded" representations of (1) the concept labels, (2) of their position in the overall ontology, and (3) as a function of all other concepts and positions within the overall ontology as it is encoded by the LRAAM.

Experimental results: In Chapters 8 and 9, we conducted and discussed experiments with the implemented LRAAM. We kept the experiments simple as they served us as a proof of concept. The results were inconclusive. An introductory experiment showed significance of a correct classification under certain conditions. Further experiments with richer input data, thereagainst, did not yield the expected results. We identified shortcomings of, and suggested possible improvements to the current implementation. These include stabilizing the LRAAM's steady state through parameter pruning, respectively the introduction of new parameters; or merging LRAAM and Deep Learning.

Research questions and contributions Our assumption is that intelligent SCC needs to understand the meaning of services to address at the same time automation and flexibility. Under this angle, we explored the realm of ontologies and pertaining techniques. By analysing the pertaining body of knowledge, we concluded that they are, in isolation, not appropriate (cf., Sect. 3.4). The reason is the lack of true understanding due to the restriction imposed by the symbol paradigm, and no "inner" and "own" representation of *meaning*. Unknown conditions can thus only be processed to the extent they are predefined. To us, this is clearly a contradiction.

However, ontologies still served our goal of more intelligent service selection and composition. They are part of a rather well-established domain one one can draw from (e.g., formalized, i.e., validated, knowledge representations). We used them to describe services and requirements. Furthermore, we defined key ontological terms. The latter provided invariants or anchor concepts to facilitate subsequent processing (cf., Sect. 4.1).

Where our goal reached beyond the ontology domain's capabilities, we explored the LRAAM. It was needed to create the above mentioned "own," i.e., non-arbitrary representations, specifically of services and requirements. These representations were used for comparison. However, the current implementation did not yield the desired results. LRAAM in its currently implemented form is not a reliable alternative matching technique as part of an intelligent SSC (cf., Sect. 9.3). Other than conceptually suggested, it does not yet yield a stable steady state adequately representing the expected (i.e, supervised) output. A crucial open question is thus how to better control the LRAAM's behavior. Specifically, how can we create a reliable, stable steady state? A possible answer is given below.

Final thoughts: Notwithstanding the results of our experimentations, we think that purely symbolic approaches to automatic integration are too restrictive when independent third-party services are concerned. Flexibility without human intervention is beyond those approaches' capabilities. Encouraged by recent developments in the area of connectionism (i.e., Deep Learning), we are however still convinced of the chosen research venue. We strongly advocate further studies, for example with a combined Deep Learning-LRAAM matching method. We are convinced that such an approach would resolve many encountered problems, including the most prominent one, namely the volatility of the network's steady state (LeCun *et al.* 15). In Figure 10.1, a possibility is sketched. Instead of initializing the label part of LRAAM's input vector with binary information, one could use the Deep Learning approach to preprocess those concept labels.

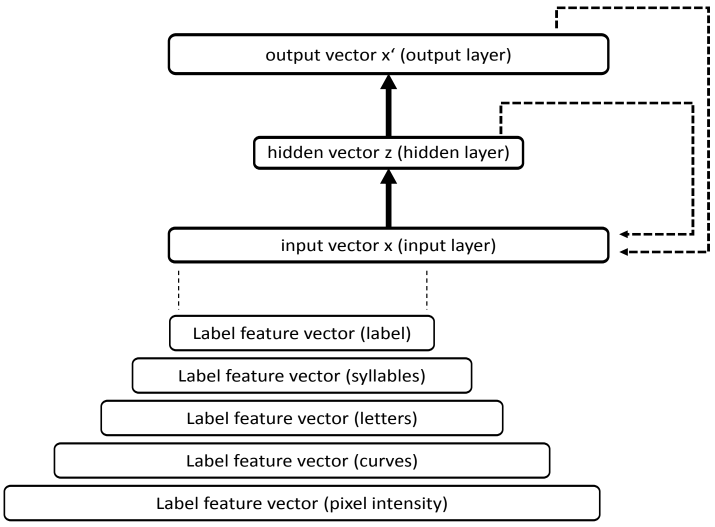

output vector x′ (output layer)
hidden vector z (hidden layer)
input vector x (input layer)
Label feature vector (label)
Label feature vector (syllables)
Label feature vector (letters)
Label feature vector (curves)
Label feature vector (pixel intensity)

Figure 10.1: Deep Learning (DL) and LRAAM. Concept labels are preprocessed by DL before participating in the input vector x.

Eventually, we concur with (Bottou 14) and (LeCun *et al.* 15) for whom new paradigms are needed to replace rule-based manipulation of symbolic expressions by operations on large vectors.

References

[Al-Said and Abdallah 09] Al-Said, G. and M. Abdallah, "An arabic text-to-speech system based on artificial neural networks.," *Journal of Computer Science*, 5(3), pp. 207–213, 2009. 148

[Allemang and Hendler 08] Allemang, D. and J. Hendler, *Semantic web for the working ontologist: Effective modeling in RDFS and OWL*, Morgan Kaufmann, May 2008. 62

[Antoniou and van Harmelen 08] Antoniou, G. and F. van Harmelen, *A semantic web primer*, The MIT Press, Cambridge Massachusetts, 2 edition, 2008. 7, 56, 58, 64, 89

[Antoniou and van Harmelen 09] Antoniou, G. and F. van Harmelen, *Web ontology language: OWL*, in *Handbook on ontologies*, Staab, S. and R. Studer, editors, pp. 91–110, Springer, 2009. 61, 62, 64, 89

[Atencia *et al.* 11] Atencia, M., J. Euzenat, G. Pirró and M.-C. Rousset, *Alignment-based trust for resource finding in semantic p2p networks*, in *The semantic web–ISWC 2011*, pp. 51–66, Springer, 2011. 112

[Baader *et al.* 09] Baader, F., I. Horrocks and U. Sattler, *Description logics*, in *Handbook on ontologies*, Staab, S. and R. Studer, editors, pp. 21–43, Springer, 2009. 9

[Babu and Darsi 13] Babu, D. V. and M. P. Darsi, "A survey on service oriented architecture and metrics to measure coupling," *International Journal on Computer Science & Engineering*, 5, pp. 726–733, 2013. 41

[Bagosi *et al.* 14] Bagosi, T., D. Calvanese, J. Hardi, S. Komla-Ebri, D. Lanti, M. Rezk, M. Rodríguez-Muro, M. Slusnys and G. Xiao, *The ontop framework for ontology based data access*, in *The semantic web and web science*, pp. 67–77, Springer, 2014. 9

[Bagui 09] Bagui, S., "Mapping owl to the entity relationship and extended entity relationship models," *Int. J. Knowl. Web Intell.*, 1, pp. 125–149, August 2009. 9

[Berners-Lee *et al.* 01] Berners-Lee, T., J. Hendler, O. Lassila *et al.*, "The semantic web," *Scientific american*, 284(5), pp. 28–37, 2001. 57

[Beynon-Davies 03] Beynon-Davies, P., *Database systems*, Macmillan, 3 edition, 2003. 25

[Blank *et al.* 92] Blank, D., L. A. Meeden and J. B. Marshall, "Exploring the symbolic/subsymbolic continuum: A case study of RAAM," in *The symbolic and connectionist paradigms: Closing the gap*, pp. 113–148, Erlbaum, 1992. 89, 116, 117, 118

[Borgo and Masolo 10] Borgo, S. and C. Masolo, *Ontological foundations of DOLCE*, in *Theory and applications of ontology: Computer applications*, pp. 279–295, Springer, 2010. 65

REFERENCES

[Born *et al.* 07] Born, M., C. Drumm, I. Markovic and I. Weber, "SUPER - raising business process management back to the business level," *ERCIM News*, 2007(70), 2007. 6, 7, 151

[Bottou 14] Bottou, L., "From machine learning to machine reasoning," *Machine learning*, 94(2), pp. 133–149, 2014. 155

[Britannica 15] Britannica, E., *Semantics*, `www.britannica.com/science/semantics`, 2015, consulted July 26, 2015. 56

[Brunsson 02] Brunsson, N., *The organisation of hyprocrisy*, Copenhagen Business School Press, 2002. 4

[Bughin and Chui 10] Bughin, J. and M. Chui, "The rise of the networked enterprise: Web 2.0 finds its payday," *McKinsey quarterly*, 4, pp. 3–8, 2010. 2, 21

[Burstein *et al.* 04] Burstein, M., J. Hobbs, O. Lassila, D. Mcdermott, S. Mcilraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin *et al.*, "OWL-S: Semantic markup for web services," *W3C Member Submission*, 2004. 71, 72, 75

[Callon 90] Callon, M., "Techno-economic networks and irreversibility," *The Sociological Review*, 38(S1), pp. 132–161, 1990. 15

[Cardoso and Sheth 05] Cardoso, J. and A. Sheth, *Introduction to semantic web services and web process composition*, in *Semantic web services and web process composition*, pp. 1–13, Springer, 2005. 55

[Chan 03] Chan, S. W. K., "Dynamic context generation for natural language understanding: A multifaceted knowledge approach," *IEEE Transactions on systems, man, and Cybernetics - Part A: Systems and Humans*, 33(1), pp. 23–41, Jan 2003. 8, 116

[Chandler 02] Chandler, D., *Semiotics: The basics*, Routledge, NY, 2002. 25

[Chen 13] Chen, D., "Framework for enterprise interoperability and maturity model (CEN/ISO 11354)," *Interoperability for enterprise software and applications*, pp. 15–22, 2013. 21

[Chen and Whalley 11] Chen, H. and J. Whalley, *The WTO government procurement agreement and its impacts on trade*, Technical report, National Bureau of Economic Research, 2011. 16

[Chen *et al.* 08] Chen, D., G. Doumeingts and F. Vernadat, "Architectures for enterprise integration and interoperability: Past, present and future," *Computers in industry*, 59(7), pp. 647–659, 2008. 37

[Cherry 67] Cherry, H., *On human communication*, MIT Press, Cambridge, Massachusetts, 1967. 56

[Cisco 14] Cisco, *Cisco global cloud index: Forecast and methodology: 2013 - 2018*, White paper, Cisco Systems Inc., 2014. 2

[Committee 90] Committee, I. C. S. S. C., *IEEE standard computer dictionary: A compilation of IEEE standard computer glossaries, 610*, ANSI / IEEE Std, IEEE, 1990. 21

[Cruz-Cunha 09] Cruz-Cunha, M. M., *Enterprise information systems for business integration in SMEs: Technological, organizational, and social dimensions: Technological, organizational, and social dimensions*, IGI Global, 2009. 41

[DARPA 08] DARPA, *DARPA-BAA-09-03 machine reading broad agency announcement (BAA)*, Online, 11 2008. 10

[d'Avila Garcez *et al.* 09] d'Avila Garcez, A. S., L. C. Lamb and D. M. Gabbay, *Neural-symbolic cognitive reasoning*, Springer, 2009. 8

[De Bruijn *et al.* 09] De Bruijn, J., M. Kerrigan, M. Zaremba and D. Fensel, *Semantic web services*, in *Handbook on ontologies*, pp. 617–636, Springer, 2009. 80, 81, 83, 84

[de Gerlachey *et al.* 94] de Gerlachey, M., A. Sperdutiz and A. Staritaz, "Using Labeling RAAM to encode medical conceptual graphs," *NNESMED'94 proceedings*, 1994. 8, 88, 119

[Diallo 14] Diallo, G., "An effective method of large scale ontology matching," *Journal of Biomedical Semantics*, 5, 2014. 10

[Do and Rahm 02] Do, H.-H. and E. Rahm, "COMA: a system for flexible combination of schema matching approaches," in *Proceedings of the 28th international conference on very large data bases*, pp. 610–621, VLDB Endowment, 2002. 10

[Dodig-Crnkovic 02] Dodig-Crnkovic, G., "Scientific methods in computer science," in *Conference for the promotion of research in it at new universities and at university colleges in sweden*, April 2002. 12

[Domingos *et al.* 13] Domingos, D., R. Martinho and C. Cândido, "Flexibility in cross-organizational ws-bpel business processes," *Procedia Technology*, 9, pp. 584–595, 2013. 47

[Eder and Wiggisser 07] Eder, J. and K. Wiggisser, "Detecting changes in ontologies via DAG comparison," in *Lecture notes in computer science 4495*, pp. 21–35, 2007. 122

[El-Sheikh *et al.* 13] El-Sheikh, E., S. Bagui, D. Firesmith, I. Petrov, N. Wilde and A. Zimmermann, "Towards semantic-supported smartlife system architectures for big data services in the cloud," in *Proceedings of the fifth international conference on advanced service computing (service computation 2013), may*, pp. 59–64, 2013. 67

[Ellingsen 97] Ellingsen, B. K., *Distributed representations of object-oriented specifications for analogical mapping*, Technical report, Citeseer, 1997. 119, 122, 125, 142, 148

[Erl 04] Erl, T., *Service-oriented architecture: a field guide to integrating XML and web services*, Prentice Hall PTR, 2004. 20

[Erl *et al.* 14] Erl, T., P. Chelliah, C. Gee, J. Kress, B. Maier, H. Normann, L. Shuster, B. Trops, C. Utschig, P. Wik and T. Winterberg, *Next generation SOA: A concise introduction to service technology & service-orientation*, The Prentice Hall Service Technology Series from Thomas Erl, Pearson Education, 2014. 20

[Euzenat and Shvaiko 07] Euzenat, J. and P. Shvaiko, *Ontology matching*, Springer, 2007. 9, 111, 112

[Euzenat and Shvaiko 13] Euzenat, J. and P. Shvaiko, *Ontology matching*, Springer, 2 edition, 2013. 7, 112, 113, 120, 121

[Fagin *et al.* 09] Fagin, R., L. M. Haas, M. Hernández, R. J. Miller, L. Popa and Y. Velegrakis, *Schema mapping creation and data exchange*, in *Conceptual modeling: Foundations and applications*, pp. 198–236, Springer, 2009. 9

[Farrag *et al.* 13] Farrag, T. A., A. I. Saleh and H. A. Ali, "Toward SWSs discovery: mapping from wsdl to owl-s based on ontology search and standardization engine," *Knowledge and Data Engineering, IEEE Transactions on*, 25(5), pp. 1135–1147, 2013. 73, 74

[Fasbinder 08] Fasbinder, M., *BPEL or ESB: Which should you use?*, www.ibm.com/developerswork/websphere/library/techarticles/0803_fasbinder2/0803_fasbinder2.html, March 2008, consulted July 16, 2015. 40

[Fenner 03] Fenner, J., *Enterprise application integration techniques*, 2003. 23

[Fensel and Bussler 02] Fensel, D. and C. Bussler, "The web service modeling framework WSMF," *Electronic Commerce Research and Applications*, 1(2), pp. 113–137, 2002. 2, 35, 39, 55

[Fensel *et al.* 11] Fensel, D., F. M. Facca, E. Simperl and I. Toma, *Semantic web services*, Springer Science & Business Media, 2011. 6, 17, 55, 57, 68, 69, 71, 74, 75, 77, 81, 84, 85, 151

[Ferreira 13] Ferreira, D. R., *Enterprise systems integration: A process-oriented approach*, Springer Science & Business Media, 2013. 32, 33, 34, 42

[Fonseca *et al.* 00] Fonseca, F. T., M. J. Egenhofer, C. Davis and K. A. Borges, "Ontologies and knowledge sharing in urban GIS," *Computers, Environment and Urban Systems*, 24(3), pp. 251–272, 2000. 66

[Fonseca *et al.* 03] Fonseca, F., C. Davis and G. Câmara, "Bridging ontologies and conceptual schemas in geographic information integration," *GeoInformatica*, 7(4), pp. 355–378, 2003. 65

[Frank and Cartwright 13] Frank, R. and E. Cartwright, *Microeconomics and behaviour*, McGraw Hill, 2013. 2

[Geebelen *et al.* 08] Geebelen, K., S. Michiels and W. Joosen, "Dynamic reconfiguration using template based web service composition," in *Proceedings of the 3rd workshop on middleware for service oriented computing*, pp. 49–54, ACM, 2008. 48, 50

[Genesereth and Nilsson 87] Genesereth, M. R. and N. J. Nilsson, "Logical foundations of artificial," *Intelligence. Morgan Kaufmann*, 1987. 58

[Gonen *et al.* 15] Gonen, B., X. Fang, E. El-Sheikh, S. Bagui, N. Wilde and A. Zimmermann, "Ontological support for the evolution of future services oriented architectures," *Transactions on Machine Learning and Artificial Intelligence*, 2(6), 2015. 67

[Guarino 98] Guarino, N., *Formal ontology in information systems: Proceedings of the first international conference (fois'98), june 6-8, trento, italy*, vol. 46, IOS press, 1998. 64

[Guarino *et al.* 09] Guarino, N., D. Oberle and S. Staab, *What is an ontology*, in *Handbook on ontologies*, Staab, S. and R. Studer, editors, pp. 1–17, Springer, 2009. 58

[Gupta and Sharma 03] Gupta, J. and S. K. Sharma, editors, *Intelligent enterprises of the 21st century*, IGI Global, Hershey, PA, USA, 2003. 19

[Habermas 84] Habermas, J., *The theory of communicative action*, Beacon Press, Boston, 1984. 24

[Hammer and Hitzler 07] Hammer, B. and P. Hitzler, *Perspectives of neural-symbolic integration*, Springer, 2007. 8

[Hanseth and Ciborra 07] Hanseth, O. and C. Ciborra, *Risk, complexity and ICT*, Edward Elgar Publishing, 2007. 15

[Harnad 03] Harnad, S., "Categorical perception," *Encyclopedia of Cognitive Science.*, 2003. 56

[Hasselbring 00] Hasselbring, W., "Information system integration," *Communications of the ACM*, 43(6), pp. 32–38, 2000. 22

[He and Da Xu 14] He, W. and L. Da Xu, "Integration of distributed enterprise applications: a survey," *Industrial Informatics, IEEE Transactions on*, 10(1), pp. 35–42, 2014. 38

[Hedman and Andersson 14] Hedman, J. and B. Andersson, "Selection method for COTS systems," *Procedia Technology*, 16, pp. 301–309, 2014. 17

[Hinton *et al.* 86] Hinton, G., J. McClelland and D. Rumelhart, "Distributed representations," in *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1*, pp. 77–109, MIT Press, 1986. 8

[Hitzler *et al.* 05] Hitzler, P., S. Bader and A. Garcez, "Ontology learning as a use-case for neural-symbolic integration," in *In proceedings of the ijcai-05 workshop on neural-symbolic learning and reasoning, nesy 05*, 2005. 10

[Hüner *et al.* 09] Hüner, K. M., M. Ofner and B. Otto, "Towards a maturity model for corporate data quality management," in *Proceedings of the 2009 acm symposium on applied computing*, pp. 231–238, ACM, 2009. 4

[Hoang and Le 09] Hoang, H. H. and M. T. Le, "BizKB: A conceptual framework for dynamic cross-enterprise collaboration.," in *Iccci*, Nguyen, N. T., R. Kowalczyk and S.-M. Chen, editors, vol. 5796, Lecture notes in computer science, pp. 401–412, Springer, 2009. 6, 85

[Hoang *et al.* 14] Hoang, H. H., J. J. Jung and C. P. Tran, "Ontology-based approaches for cross-enterprise collaboration: A literature review on semantic business process management," *Enterprise Information Systems*, 8(6), pp. 648–664, November 2014. 6, 85

[Hurley 03] Hurley, P., *A concise introduction to logic*, Wadsworth, Inc, 8 edition, 2003. 9

[Hussain and Mastan 14] Hussain, M. A. and M. Mastan, "A study on semantic web services and its significant trends," *IJCER*, 3(5), pp. 234–237, 2014. 2

[Int 08] *IAITAM best practice library - IBPL*, International Association of Information Technology Asset Managers, 2008. 1

[IT 07] *COBIT 4.1*, The IT Governance Institute, 2007. 2

[ITS 07] *It service management based on ITIL v3 - a pocket guide*, ITSMF International, 2007. 1

[Izza 09] Izza, S., "Integration of industrial information systems: from syntactic to semantic integration approaches," *Enterprise Information Systems*, 3(1), pp. 1–57, 2009. 1, 3, 6, 22, 23, 24, 84, 85

[Izza *et al.* 08] Izza, S., L. Vincent and P. Burlat, "Exploiting semantic web services in achieving flexible application integration in the microelectronics field," *Computers in industry*, 59(7), pp. 722–740, 2008. 79

[Jardim-Goncalves *et al.* 13] Jardim-Goncalves, R., A. Grilo, C. Agostinho, F. Lampathaki and Y. Charalabidis, "Systematisation of interoperability body of knowledge: The foundation for enterprise interoperability as a science," *Enterprise Information Systems*, 7(1), pp. 7–32, 2013. 9

[Jones 08] Jones, T. M., *Artificial intelligence - a systems approach*, Infinity Science Press, 2008. 10

[Jrad *et al.* 15] Jrad, F., J. Tao, A. Streit, R. Knapper and C. Flath, "A utility-based approach for customised cloud service selection," *International Journal of Computational Science and Engineering*, 10(1/2), pp. 32–44, 2015. 17

[Juric 07] Juric, M. B., *SOA approach to integration: XML, web services, ESB, and BPEL in real-world SOA projects*, Packt Publishing Ltd, 2007. 28, 29, 32, 41

[Kaiya and Saeki 06] Kaiya, H. and M. Saeki, "Using domain ontology as domain knowledge for requirements elicitation," in *Requirements engineering, 14th ieee international conference*, pp. 189–198, IEEE, 2006. 66

[Kale 14] Kale, V., *Guide to cloud computing for business and technology managers: From distributed computing to cloudware applications*, Taylor & Francis, 2014. 2, 20, 33, 38

[Kamaruddin *et al.* 12] Kamaruddin, L. A., J. Shen and G. Beydoun, "Evaluating usage of WSMO and OWL-S in semantic web services," in *Proceedings of the eighth asia-pacific conference on conceptual modelling-volume 130*, pp. 53–58, Australian Computer Society, Inc., 2012. 80, 84, 152

[Kapuruge *et al.* 11] Kapuruge, M., J. Han and A. Colman, "Controlled flexibility in business processes defined for service compositions," in *Services computing (scc), 2011 ieee international conference on*, pp. 346–353, IEEE, 2011. 45, 46, 47

[Kim *et al.* 12] Kim, J.-P., J.-E. Hong, J.-Y. Choi and Y.-H. Cho, "Dynamic service orchestration for SaaS application in web environment," in *Proceedings of the 6th international conference on ubiquitous information management and communication*, p. 43, ACM, 2012. 49, 50

[Klusch and Kapahnke 12] Klusch, M. and P. Kapahnke, "The iSeM matchmaker: A flexible approach for adaptive hybrid semantic service selection," *Web Semantics: Science, Services and Agents on the World Wide Web*, 15, pp. 1–14, 2012. 17, 18

[Klusch *et al.* 09] Klusch, M., B. Fries and K. Sycara, "OWLS-MX: A hybrid semantic web service matchmaker for OWL-S services," *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(2), pp. 121–133, 2009. 79

[Koning *et al.* 09] Koning, M., C.-a. Sun, M. Sinnema and P. Avgeriou, "VxBPEL: Supporting variability for web services in bpel," *Information and Software Technology*, 51(2), pp. 258–269, 2009. 47

[Kontchakov *et al.* 13] Kontchakov, R., M. Rodríguez-Muro and M. Zakharyaschev, *Ontology-based data access with databases: A short course*, in *Reasoning web. semantic technologies for intelligent data access*, pp. 194–229, Springer, 2013. 9

[Kotis *et al.* 06] Kotis, K., G. Vouros and K. Stergiou, "Towards automatic merging of domain ontologies: The HCONE-merge approach," *Journal of Web Semantics (JWS)*, 4, pp. 60–79, 2006. 112

[Kourtesis and Paraskakis 10] Kourtesis, D. and I. Paraskakis, *Supporting semantically enhanced web service discovery for enterprise application integration*, in *Semantic enterprise application integration for business processes: Service-oriented frameworks: Service-oriented frameworks*, Mentzas, G., editor, chapter 6, pp. 105–130, IGI Global, 2010. 67

[Koutra *et al.* 11] Koutra, D., A. Parikh, A. Ramdas and J. Xiang, *Algorithms for graph similarity and subgraph matching*, Technical report, Technical Report of Carnegie-Mellon-University, 2011. 121

[Kress *et al.* 13] Kress, J., B. Maier, H. Normann, D. Schmeidel, G. Schmutz, B. Trops and T. W. Utschig, *Enterprise service bus*, 2013. 38, 40

[Krizevnik and Juric 12] Krizevnik, M. and M. B. Juric, "Data-bound variables for WS-BPEL executable processes," *Computer Languages, Systems and Structures*, 38(4), pp. 279–299, December 2012. 47

[Lausen and Farrell 07] Lausen, H. and J. Farrell, "Semantic annotations for WSDL and XML schema," *W3C recommendation, W3C*, 2007. 69

[LeCun *et al.* 15] LeCun, Y., Y. Bengio and G. Hinton, "Deep learning," *Nature*, 521(7553), pp. 436–444, May 2015. 148, 150, 152, 154, 155

[Lee *et al.* 03] Lee, J., K. Siau and S. Hong, "Enterprise integration with ERP and EAI," *Commun. ACM*, 46(2), pp. 54–60, February 2003. 20

[Lehaney *et al.* 11] Lehaney, B., P. Lovett and M. Shah, *Business information systems and technology: a primer*, Routledge, 2011. 2, 27

[Lemcke 10] Lemcke, J., *Scalable ontological EAI and e-business integration*, KIT Scientific Publishing, 2010. 69, 80

[Lewis *et al.* 08] Lewis, G., D. B. Smith *et al.*, "Service-oriented architecture and its implications for software maintenance and evolution," in *Frontiers of software maintenance*, pp. 1–10, IEEE, 2008. 67

[Lheureux 12] Lheureux, B. J., *Predicts 2013: Application integration*, Technical report, Gartner, 2012. 5

[Li *et al.* 12] Li, W., R. Raskin and M. F. Goodchild, "Semantic similarity measurement based on knowledge mining: An artificial neural net approach," *International Journal of Geographical Information Science*, 26(8), pp. 1415–1435, 2012. 118

[Lin 98] Lin, D., "An information-theoretic definition of similarity.," in *Icml*, vol. 98, pp. 296–304, 1998. 120

[Linthicum 99] Linthicum, D. S., *Enterprise application integration*, Information Technology, Addison-Wesley, 1999. 5, 22, 23

[Linthicum 03] Linthicum, D. S., *Next generation application integration: from simple information to web services*, Addison-Wesley Longman Publishing Co., Inc., 2003. 22

[Liu *et al.* 07] Liu, A., Q. Li, L. Huang and M. Xiao, "A declarative approach to enhancing the reliability of BPEL processes," in *Web services, 2007. icws 2007. ieee international conference on*, pp. 272–279, IEEE, 2007. 47

[Liu *et al.* 15] Liu, S., W. Li, K. Liu and J. Han, *Evaluation frameworks for information systems integration: from a semiotic lens*, in *Liss 2013*, pp. 1333–1340, Springer, 2015. 25

REFERENCES

[Lubblinsky and Tyomkin 03] Lubblinsky, B. and D. Tyomkin, “Dissecting service oriented architectures,” *Business Integration Journal*, 5(10), pp. 52–58, 2003. 23

[Ludolph 04] Ludolph, H., *A communication model towards flexible associations of business entities within evolving environments*, Master Thesis, HEC Montreal, 2004. 24

[Ludolph *et al.* 11] Ludolph, H., P. Kropf and G. Babin, “SoftwIre integration - an onto-neural perspective,” in *E-technologies: Transformation in a connected world - 5th international conference (mcetech 2011). les diablerets, switzerland, january 23-26, 2011, revised selected papers*, Babin, G., K. Stanoevska-Slabeva and P. Kropf, editors, numéro 78, Lecture notes in business information processing, pp. 116–130, Springer, May 2011. 90

[Makhoul *et al.* 99] Makhoul, J., F. Kubala, R. Schwartz, R. Weischedel *et al.*, “Performance measures for information extraction,” in *Proceedings of darpa broadcast news workshop*, pp. 249–252, 1999. 10

[Martins and de Almeida Falbo 08] Martins, A. F. and R. de Almeida Falbo, “Models for representing task ontologies.,” in *Wonto*, 2008. 66

[Mizoguchi *et al.* 95] Mizoguchi, R., J. Vanwelkenhuysen and M. Ikeda, “Task ontology for reuse of problem solving knowledge,” *Towards Very Large Knowledge Bases: Knowledge Building & Knowledge Sharing*, pp. 46–59, 1995. 66

[Modica and Tomarchio 15] Modica, G. D. and O. Tomarchio, “A semantic framework to support resource discovery in future cloud markets,” *International Journal of Computational Science and Engineering*, 10(1-2), pp. 1–14, 2015. 17

[Mohamed *et al.* 07] Mohamed, A., G. Ruhe and A. Eberlein, “COTS selection: past, present, and future,” in *Engineering of computer-based systems, 2007. ecbs’07. 14th annual ieee international conference and workshops on the*, pp. 103–114, IEEE, 2007. 5, 16, 17

[Molina *et al.* 98] Molina, A., J. M. Sánchez and A. Kusiak, *Handbook of life cycle engineering: concepts, models and technologies*, Springer, 1998. 19

[Motik 05] Motik, B., “On the properties of metamodeling in OWL,” in *In 4th int. semantic web conf. (iswc 2005*, pp. 548–562, 2005. 62

[Nassif and Capretz 13] Nassif, A. B. and M. A. Capretz, *Offering SaaS as SOA services*, in *Innovations and advances in computer, information, systems sciences, and engineering*, pp. 405–414, Springer, 2013. 52

[Newmarch 12] Newmarch, J., *Network programming with go*, jan.newmarch.name/go/. 29

[Niwattanakul *et al.* 07] Niwattanakul, S., P. Martin, M. Eboueya and K. Khaimook, “Ontology mapping based on similarity measure and fuzzy logic,” in *Proceedings of world conference on e-learning in corporate, government, healthcare, and higher education*, Richards, G., editor, pp. 6383–6387, Quebec City, Canada, 2007. 58

[Oberle *et al.* 05] Oberle, D., S. Lamparter, A. Eberhart, S. Grimm, S. Agarwal, R. Studer and P. Hitzler, “Semantic management of web services using the core ontology of services,” *Workshop paper*, 2005. 87

[Obrst 03] Obrst, L., “Ontologies for semantically interoperable systems,” in *Proceedings of the twelfth international conference on information and knowledge management*, pp. 366–369, ACM, 2003. 55

[Obrst 10] Obrst, L., *Ontological architectures*, in *Theory and applications of ontology: Computer applications*, pp. 27–66, Springer, 2010. 66

[Olson 82] Olson, M., *The rise and decline of nations*, Yale University Press, 1982. 4

[Otero-Cerdeira *et al.* 15] Otero-Cerdeira, L., F. J. Rodríguez-Martínez and A. Gómez-Rodríguez, "Ontology matching: A literature review," *Expert Systems with Applications*, 42(2), pp. 949–971, 2015. 10

[Pan 09] Pan, J. Z., *Resource description framework*, in *Handbook on ontologies*, Staab, S. and R. Studer, editors, pp. 71–90, Springer, 2009. 64

[Panetto 07] Panetto, H., "Towards a classification framework for interoperability of enterprise applications," *International Journal of Computer Integrated Manufacturing*, 20(8), pp. 727–740, 2007. 21

[Panetto and Cecil 13] Panetto, H. and J. Cecil, "Information systems for enterprise integration, interoperability and networking: theory and applications," *Enterprise Information Systems*, 7(1), pp. 1–6, 2013. 3, 21

[Pasley 05] Pasley, J., "How BPEL and SOA are changing web services development," *IEEE Internet Computing*, 9(3), pp. 60–67, 2005. 40, 41

[Peirce 58] Peirce, C. S., *Collected writings (8 vols.)*, Harvard University Press, Cambridge, MA, 1931-58. 25

[Poggi *et al.* 08] Poggi, A., D. Lembo, D. Calvanese, M. Lenzerini and R. Rosati, "Linking data to ontologies," *Journal on Data Semantics*, pp. 133–173, 2008. 8

[Pollack 90] Pollack, J. B., "Recursive distributed representations," *Artificial Intelligence*, 46, pp. 77–105, 1990. 119

[Prestes *et al.* 13] Prestes, E., J. L. Carbonera, S. R. Fiorini, V. A. Jorge, M. Abel, R. Madhavan, A. Locoro, P. Goncalves, M. E. Barreto, M. Habib *et al.*, "Towards a core ontology for robotics and automation," *Robotics and Autonomous Systems*, 61(11), pp. 1193–1204, 2013. 64, 66

[Qu and Buyya 14] Qu, C. and R. Buyya, "A cloud trust evaluation system using hierarchical fuzzy inference system for service selection," in *Advanced information networking and applications (aina), 2014 ieee 28th international conference on*, pp. 850–857, IEEE, 2014. 17

[Qu *et al.* 13] Qu, L., Y. Wang, M. Orgun *et al.*, "Cloud service selection based on the aggregation of user feedback and quantitative performance assessment," in *Services computing (scc), 2013 ieee international conference on*, pp. 152–159, IEEE, 2013. 17

[Rahm 11] Rahm, E., *Towards large-scale schema and ontology matching*, in *Schema matching and mapping*, pp. 3–27, Springer, 2011. 10

[Rahm and Bernstein 01] Rahm, E. and P. A. Bernstein, "A survey of approaches to automatic schema matching," *The VLDB Journal*, 10(4), December 2001. 111

[Rebstock *et al.* 08] Rebstock, M., F. Janina and H. Paulheim, *Ontologies-based business integration*, Springer Science & Business Media, 2008. 61

[Regev *et al.* 07] Regev, G., I. Bider and A. Wegmann, "Defining business process flexibility with the help of invariants," *Software Process: Improvement and Practice*, 12(1), pp. 65–79, 2007. 45

[Reichert and Rinderle-Ma 06] Reichert, M. and S. Rinderle-Ma, "On design principles for realizing adaptive service flows with BPEL," in *Proc. workshop (emisa'06)*, 2006. 47

[Rijsbergen 79] Rijsbergen, C. J. V., *Information retrieval*, Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979. 10

[Rittgen 08] Rittgen, P., *Handbook of ontologies for business interaction*, IGI Global, 2008. 8

[Rodríguez *et al.* 12] Rodríguez, D., J. Hermosillo and B. Lara, "Meaning in artificial agents: The symbol grounding problem revisited," *Minds and Machines*, 22(1), pp. 25–34, 2012. 56

[Roshen 09] Roshen, W., *SOA-based enterprise integration: A step-by-step guide to services-based application*, McGraw-Hill, Inc., 2009. 21, 27, 28, 30, 31, 33, 37, 40, 41, 45

[Roussey *et al.* 11] Roussey, C., F. Pinet, M. A. Kang and O. Corcho, *An introduction to ontologies and ontology engineering*, in *Ontologies in urban development projects*, pp. 9–38, Springer, 2011. 64, 65, 66

[Ruhe 03] Ruhe, G., *Intelligent support for selection of COTS products*, in *Web, web-services, and database systems*, pp. 34–45, Springer, 2003. 5, 16

[SalesForce.com 15] SalesForce.com, *Selecting the right salesforce CRM edition*, `www.salesforce.com/ap/assets/pdf/cloudforce/PricingEditions-SelectingTheRightSalesforceCRMEditi` 2015, consulted July 24, 2015. 51

[Saussure 83] Saussure, F. d., *Course in general linguistics*, Duckworth, London, 1983. 25, 26

[Searle 80] Searle, J. R., "Minds, brains, and programs," *Behavioral and brain sciences*, 3(03), pp. 417–424, 1980. 56

[Sequeda *et al.* 12] Sequeda, J. F., M. Arenas and D. P. Miranker, "On directly mapping relational databases to RDF and OWL," in *Proceedings of the 21st international conference on world wide web*, pp. 649–658, New York, NY, USA, ACM, 2012. 9

[Sheng *et al.* 14] Sheng, Q. Z., X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne and X. Xu, "Web services composition: A decade's overview," *Information Sciences*, 280, pp. 218–238, 2014. 80

[Sherif 09] Sherif, M. H., *Handbook of enterprise integration*, CRC Press, 2009. 31

[Sheth 99] Sheth, A. P., *Changing focus on interoperability in information systems: from system, syntax, structure to semantics*, in *Interoperating geographic information systems*, pp. 5–29, Springer, 1999. 23

[Shin *et al.* 09] Shin, D.-H., K.-H. Lee and T. Suda, "Automated generation of composite web services based on functional semantics," *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(4), pp. 332–343, 2009. 79

[Shvaiko and Euzenat 05] Shvaiko, P. and J. Euzenat, *A survey of schema-based matching approaches*, in *Journal on data semantics iv*, pp. 146–171, Springer, 2005. 112

[Shvaiko and Euzenat 13] Shvaiko, P. and J. Euzenat, "Ontology matching: state of the art and future challenges," *IEEE Transactions on Knowledge and Data Engineering*, 25(1), pp. 158–176, 2013. 10

[Silcher *et al.* 12] Silcher, S., J. Minguez and B. Mitschang, *A novel approach to product lifecycle management based on service hierarchies*, Springer, 2012. 41

[Singh and Huhns 05] Singh, M. and M. Huhns, *Service-oriented computing: Semantics, processes, agents*, Wiley, 2005. 3, 22

[Smirnov *et al.* 03] Smirnov, A., M. Pashkin, N. Chilov, T. Levashova and A. Krizhanovsky, "Knowledge sharing for continuous business engineering based on web intelligence," in *Proceeding of the 10th international conference on concurrent engineering*, pp. 677–684, Madeira Portugal, A.A. Balkema Publishers, July 2003. 23

[Sperduti 93] Sperduti, A., *On some stability properties of the LRAAM model*, Technical report, International Computer Science Institute, 1993. 8, 88, 119, 148

[Stimmel 01] Stimmel, G., "How to choose an e-business vendor," *Office World News*, 2001. 5

[Stonebraker 99] Stonebraker, M., "Integrating islands of information," *eAI Journal*, 1(10), pp. 1–5, 1999. 22, 23

[Studer *et al.* 98] Studer, R. R., R. Benjamins and D. Fensel, "Knowledge engineering: principles and methods," *Data and knowledge engineering*, 25, pp. 161–197, 1998. 57

[Sundareswaran *et al.* 12] Sundareswaran, S., A. Squicciarini and D. Lin, "A brokerage-based approach for cloud service selection," in *Cloud computing (cloud), 2012 ieee 5th international conference on*, pp. 558–565, IEEE, 2012. 17

[Tan and Zhou 13] Tan, W. and M. Zhou, *Business and scientific workflows: A web service-oriented approach*, IEEE Press Series on Systems Science and Engineering, Wiley, 2013. 45

[Tichy 97] Tichy, W. F., "Should computer scientists experiment more? - 16 excuses to avoid experimentation," *IEEE Computer*, 31, pp. 32–40, 1997. 12

[Trinh *et al.* 06] Trinh, Q., K. Barker and R. Alhajj, "RDB2ONT: A tool for generating OWL ontologies from relational database systems," in *Proceedings of the advanced int'l conference on telecommunications and int'l conference on internet and web applications and services*, AICT-ICIW '06, pp. 170–, Washington, DC, USA, IEEE Computer Society, 2006. 9

[Ulrich 01] Ulrich, W., "A philosophical staircase for information systems definition, design, and development," *Journal of Information Technology Theory and Application*, 3(3), 2001. 24

[Van Der Aalst *et al.* 03] Van Der Aalst, W. M., A. H. Ter Hofstede and M. Weske, *Business process management: A survey*, in *Business process management*, pp. 1–12, Springer, 2003. 45, 46, 47, 79

[Vernadat 96] Vernadat, F., *Enterprise modeling and integration: Principles and applications*, Chapman and Hall, London, 1996. 19

[Vernadat 02] Vernadat, F. B., "Enterprise modeling and integration (EMI): Current status and research perspectives," *Annual Reviews in Control*, 26(1), pp. 15–25, 2002. 22

[Wang *et al.* 12] Wang, H. H., N. Gibbins, T. R. Payne and D. Redavid, "A formal model of the semantic web service ontology (WSMO)," *Information Systems*, 37(1), pp. 33–60, 2012. 81

[Wei *et al.* 11] Wei, D., T. Wang, J. Wang and A. Bernstein, "SAWSDL-iMatcher: A customizable and effective semantic web service matchmaker," *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(4), pp. 402–417, 2011. 68

[Whaiduzzaman *et al.* 14] Whaiduzzaman, M., A. Gani, N. B. Anuar, M. Shiraz, M. N. Haque and I. T. Haque, "Cloud service selection using multicriteria decision analysis," *The Scientific World Journal*, 2014, 2014. 17

[Xu *et al.* 03] Xu, Y., D. Sauquet, P. Degoulet and M.-C. Jaulent, "Component-based mediation services for the integration of medical applications," *Artificial Intelligence in Medicine*, 27(3), pp. 283–304, 2003. 23

[Xu *et al.* 06] Xu, Z., S. Zhang and Y. Dong, "Mapping between relational database schema and OWL ontology for deep annotation," in *Proceedings of the 2006 ieee/wic/acm international conference on web intelligence*, WI '06, pp. 548–552, Washington, DC, USA, IEEE Computer Society, 2006. 9

[Xu *et al.* 12] Xu, Z., Y. Ni, W. He, L. Lin and Q. Yan, "Automatic extraction of OWL ontologies from UML class diagrams: a semantics-preserving approach," *World Wide Web*, 15(5-6), pp. 517–545, 2012. 9

[Zdravković *et al.* 14] Zdravković, M., M. Trajanović and H. Panetto, "Enabling interoperability as a property of ubiquitous systems: towards the theory of interoperability-of-everything," in *4th International Conference on Information Society and Technology, ICIST 2014*, vol. 1, pp. 240–247, Kopaonik, Serbia, March 2014. 5, 67, 87

[Zheng *et al.* 13] Zheng, W., L. Zou, X. Lian, D. Wang and D. Zhao, "Graph similarity search with edit distance constraint in large graph databases," in *Proceedings of the 22nd acm international conference on conference on information & knowledge management*, pp. 1595–1600, ACM, 2013. 121

[Zuñiga *et al.* 14] Zuñiga, J. C., J. J. Pérez-Alcázar, L. A. Digiampietri and S. E. Barbin, "A loosely coupled architecture for automatic composition of web services applications," *International Journal of Metadata, Semantics and Ontologies*, 9(3), pp. 241–251, 2014. 79