Predicting the Content of Peer-to-Peer Interactions

Paolo Besana

Doctor of Philosophy

Centre for Intelligent Systems and their Applications
School of Informatics
University of Edinburgh

2009

Abstract

Software agents interact to solve tasks, the details of which need to be described in a language understandable by all the actors involved. Ontologies provide a formalism for defining both the domain of the task and the terminology used to describe it. However, finding a shared ontology has proved difficult: different institutions and developers have different needs and formalise them in different ontologies.

In a closed environment it is possible to force all the participants to share the same ontology, while in open and distributed environments ontology mapping can provide interoperability between heterogeneous interacting actors. However, conventional mapping systems focus on acquiring static information, and on mapping whole ontologies, which is infeasible in open systems.

This thesis shows a different approach to the problem of heterogeneity. It starts from the intuitive idea that when similar situations arise, similar interactions are performed. If the interactions between actors are specified in formal scripts, shared by all the participants, then when the same situation arises, the same script is used. The main hypothesis that this thesis aims to demonstrate is that by analysing different runs of these scripts it is possible to create a statistical model of the interactions, that reflect the frequency of terms in messages and of ontological relations between terms in different messages. The model is then used during a run of a known interaction to compute the probability distribution for terms in received messages. The probability distribution provides additional information, contextual to the interaction, that can be used by a traditional ontology matcher in order to improve efficiency, by reducing the comparisons to the most likely ones given the context, and possibly both recall and precision, in particular helping disambiguation.

The ability to create a model that reflects real phenomena in this sort of environment is evaluated by analysing the quality of the predictions, in particular verifying how various features of the interactions, such as their non-stationarity, affect the predictions. The actual improvements to a matcher we developed are also evaluated. The overall results are very promising, as using the predictor can lower the overall computation time for matching by ten times, while maintaining or in some cases improving recall and precision.

Acknowledgements

These years have been extremely interesting and lively. I had the opportunity to exchange ideas and learn from great people.

I would like to thank my supervisors, Dave Robertson for the help he provided and for the example he gives as a person, and Micheal Rovatsos for the help and for the good times spent together.

I would also like to thank all the people I shared my time with, and helped me in the earlier stages and kept listening to my political grumbles on the destiny of the world: thanks to Adam Barker, Thomas French, Li Guo and Jarred McGinnis (the great 4.15, now spread in UK, in strict alphabetical order).

I want to thank my partner, Luna De Ferrari, for her help and for the time we spent together, and my parents who supported me and accepted the thousand miles of separation.

Finally a great thanks to my friends back home in Milano, who kept me connected to my old world.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted to any other degree or professional qualification except as specified.

[*Paolo Besana*]

Publications

The work in this thesis is based on the following publications:

- P. Besana, D. Robertson. *How Service Choreography Statistics Reduce the Ontology Mapping Problem.* International Semantic Web Conference 2007 in Busan (Korea)
- P. Besana, D. Robertson. *Probabilistic Dialogue Models for Dynamic Ontology Mapping*. Uncertainty Reasoning for the Semantic Web workshop in ISWC'06 in Athens (GA)
- P. Besana. *A Framework for Combining Ontology and Schema Matchers with Dempster-Shafer.* Ontology Matching workshop in ISWC'06 workshop in Athens (GA)
- P. Besana, D. Robertson, M. Rovatsos. *Exploiting interaction contexts in P2P ontology mapping*. P2PKM'05 workshop in San Diego
- P. Besana, D. Robertson. *Contexts in Dynamic Ontology Mapping.* Context and Ontology: Theory, Practice and Applications workshop in AAAI'05

Related publications are:

- F. McNeill, P. Besana, J. Pane, F. Giunchiglia. Service integration through structure preserving semantic matching. *Cases on Semantic Interoperability for Information Systems Integration: Practices and Applications*. Edited by Yannis Kalfoglou.
- A. Barker, P.Besana, D. Robertson, J. Weissman. The Benefits of Service Choreography for Data-Intensive Computing, to appear in CLADE '09 workshop, HPDC'09, Berlin, Germany, 2009
- G. Trecarichi, L. Vaccari, V. Rizzi, M. Marchesi, P. Besana. OpenKnowledge at work: exploring centralized and decentralized information gathering in emergency contexts, *ISCRAM'09*, Goteborg, Sweden
- P. Besana, V. Patkar, D. Glasspool and D. Robertson. *Distributed Workflows: the OpenKnowledge experience,* SEMELS '08, Monterrey

- F. Giunchiglia, F. McNeill, M. Yatskevich, J. Pane, P. Besana, P. Shvaiko. *Approximate structure preserving semantic matching*, ODBASE'08, Monterrey

- F. Giunchiglia, M.Yatskevich, F.McNeill, P. Shvaiko, J.Pane and P.Besana. *Approximate structure preserving semantic matching*, short paper, ECAI'08, Patras

- D. Robertson, C. Walton, A. Barker, P. Besana, Y. Chen-Burger, F. Hassan, D. Lambert, G. Li, J . McGinnis, N. Osman, A. Bundy, F. McNeill, F. van Harmelen, C. Sierra, F. Giunchiglia. *Models of Interaction as a Grounding for Peer to Peer Knowledge Sharing*. In E. Chang, T. Dillon, R. Meersman and K. Sycara editors, Advances in Web Semantics, vol 1, LNCS-IFIP

Contents

List of Figures

List of Tables

Chapter 1

Introduction

One of the aims of information technology is to automate repetitive or time-consuming tasks, such as numerical computations or data storage and retrieval. When tasks become more complex they often require the interaction between different actors. An interaction involves exchange of information between the actors, obtained by exchanging messages. Messages convey meanings encoded into signs for transmission: in order to understand a message, a receiver should be able to map the signs in the messages to meanings aligned with those intended by the transmitter.

Therefore, actors should agree on the signs, or terms, used to describe the domain of the interaction: for example, if an agent wants to buy a particular product from a seller, it must be able to specify the properties of the product unambiguously. In computer science, ontologies are used with this goal. An ontology is intended as the formal conceptualisation of a domain [27], expressed in a machine processable language, and it is usually the result of an agreement of experts of the domain. Sharing the same ontology is assumed in most of the web service composition frameworks and it is enforced in a multi-agent interaction framework such as Electronic Institutions [57].

A shared ontology can be a strong assumption in an open environment, as agents may come from different backgrounds and have different ontologies, designed for their specific needs. In this kind of environment, communication implies translation. The usual approach is to create an alignment between the ontologies using an ontology matcher [16], creating a sort of bilingual dictionary. Depending on the approach, matchers may compare labels or ontology structures, or may use external dictionaries like WordNet to prove similarity between nodes in hierarchies, or may learn how instances are classified to find similarities between concepts, or combine information

from different sources and so on. In open systems, the identities of the participants in interactions may be unknown until the interaction is started, and therefore matching in advance may be unfeasible. Matching during the interactions may be computationally difficult, as many interactions with different actors can take place simultaneously.

As we will see in Chapter 2 and then in more detail in Chapter 6, most available ontology mapping systems focus on acquiring *static*, *a priori* information about ontology correspondences, and aim at the widest possible ontological commitment between the ontologies. However, open systems need to minimise the ontological commitment required by participating actors. This can be obtained reducing the portions of ontologies that need to be match to those that are required for the interaction. In order to do so, the work presented in this thesis takes a different approach to the problem of semantic heterogeneity, and focusses on the messages exchanged between the actors of an interaction, with the aim of predicting their content. The predictions can then be used by an ontology matcher to improve the matching process.

This approach moves from the intuitive idea that interactions follow conventions and patterns, and these patterns are repeated when similar situations arise. For example, the brief talk between a customer and a waiter at the counter of a cafe will always be similar: a “one coffee, please” request can be followed by a “black or white?” or “espresso?” offer, but unlikely by a “It’s 2 o’clock” answer. Being in the cafe provides the context that bounds the possible set of interactions. This is a common experience, and the context in which an interaction takes place helps also when abroad, allowing us to guess the likely requests in specific occasions, even though they are pronounced in an unknown language.

Extending the example, we can imagine that we possess an “oracle” that can tell if words in two languages have the same meaning. The oracle is not a dictionary, where we can look up the direct translation of a word, but it can tell if a word in a foreign language corresponds to a specific word in our language. Without any knowledge of the context, in order to translate the request from the waiter we have to list all the words in our language until we encounter the right one. If, on the other hand, we know the context, we can select a much smaller set of terms that are usually used in such conversations, reducing the time it takes to translate.

The first and fundamental hypothesis that this thesis aims to verify is that the history of similar interactions between actors, together with the state of a running interaction, can be used to predict the content of messages in the current interaction. One of the first questions to answer is what is meant by “similar interactions”. If every

time a coordinated activity is required actors have to plan and create their interactions, as described for example in the BDI model presented in Chapter 6, recognising that one is in the same interaction becomes a difficult task. The participating actors are potentially different in every interaction, and their specific plans, that is the sequences of messages that they expect and send, may be different every time. Therefore, the requirement that I make is that actors use shared scripts to describe the interactions they are performing. The scripts define the interactions in their entirety, so that all participants follow the same protocol. Such scripts are called "choreographies", as their perspective is global and do not focus on the behaviour of a single participant. When the same situation arises, the actors choose the same choreography. The choreography forms the kernel of the interaction context and provides the boundaries of what to expect in the exchanged messages: it is a stable reference framework throughout all the repeated runs.

If the main hypothesis is correct and it is therefore possible to predict the content of messages given the history of choreography runs and the state of a current interaction, then another hypothesis follows, whose verification allows us to tackle the problem of semantic heterogeneity between agents. The idea is that, once we have reliable predictions for the content of the exchanged messages, the predictions can be used to improve dynamic ontology matching by focussing only on the portions of ontologies that are relevant to a specific interaction between actors instead of trying to match whole ontologies out of context. By excluding unrelated portions of ontologies, the use of predictions should improve efficiency, while maintaining or improving the quality, expressed in the standard measures of precision and recall, of the alignment.

To prove the hypotheses, I have created a framework based on choreographies, and I have represented the predictions of the content of a message as a probability distributions over all the possible terms in the actor's ontology. The predictions are passed to an ontology matcher, which uses them to improve its results. The model of the interaction content is created and updated by feeding back into the predictor the result of the matching process. Figure 1.1 shows the feed-back loop that forms the foundation of the predictor architecture. While the idea of agents sharing choregraphies for their interactions was initially a leap of faith, the OpenKnowledge[1] project provided a grounded example of such an architecture.

If all actors use the same ontology, it is possible to verify if the main hypothesis is correct: we can compare directly the content of messages with predictions. If the

[1] `www.openk.org`

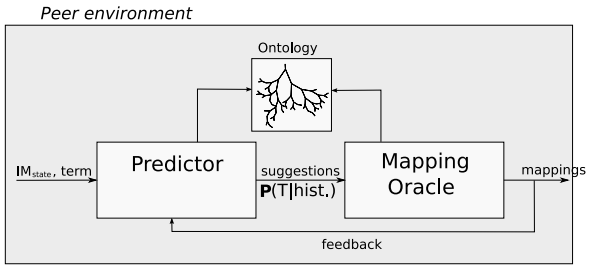

Figure 1.1: *Predictor model*

actors use different ontologies, the predictions are passed to a matcher, and if the correspondences between the ontologies are known in advance, it is possible to compare the results of the matcher, in terms of precision and recall, and evaluate the improvement brought by the predictor against the baseline provided by the ontology matcher alone.

1.1 Objectives

As we have seen above, distributed systems can be closed, and force all participants to share the same ontology, or open, and allow the participants to keep their ontology. In closed systems the participants are required to maximise their ontological commitment towards the shared ontology. By contrast, open systems can work only if they require the minimal commitment from the participants, reducing therefore the adaptation that participants need to do.

To summarise, the two key goals of this thesis are:

1. improve the efficiency of an arbitrary ontology matcher,

2. maintain or improve the quality of the matcher's results

It does so by focussing on matching only the terms that are related to the specific context of the interaction, possibly improving the quality of the matching. In other words, the thesis aims at demonstrating that it is possible to use an arbitrary ontology matcher to compare only the terms predicted to be more likely according to the statistical model of the interaction and maintain or improve the quality of matching, expressed in terms of precision and recall, that would be obtained comparing all the agent's ontology. By precision we mean the number of correct correspondences, and by recall we mean the number of found correspondences out of all the existing ones. Because only a small portion of the ontology is compared every time a message is

received, the matching process should be more efficient. The overall result is to *reduce the commitment* required for interactions between peers, thereby facilitating the adoption of really open system.

As we will see in the evaluation (Chapter 5), the predictor provides reliable suggestions after a relatively few repeated interactions. These suggestions, once fed to an ontology matcher, can reduce the overall time of matching by ten times, while maintaining, and in some cases improving, recall and precision.

1.2 Contributions to knowledge

This thesis shows how the use of choreographies to coordinate interactions between agents, under the reasonable assumption that the same choreography is used when similar situations arise, can be exploited to create a statistical model of the exchanged messages in the interactions. In a sense, the statistical model is a context for agent interactions, that is created and updated analysing the history of repeated runs of the same choreography.

The agents need to recognise when they are in an interaction that has been previously encountered. Not all coordinations models for agents are fit for this purpose: only models that consider interactions as first-class objects allow this. This thesis shows how choreographies are be particularly suitable because they define interactions in their entirety, covering equally all participants.

Within a choreography, the statistical model can be used to predict the likely content of the messages, and the predictions can be used to improve the efficiency of an ontology matcher, maintaining or improving the quality of the computed correspondences.

1.3 Applications

The open environment presented in the introduction is the founding assumption of the OpenKnowledge[2] project, that will be described in more detail in Chapter 2. OpenKnowledge provides the framework for the creation of peer-to-peer communities, that is networks of peer nodes, each playing both as server and client to every other node in the network. In OpenKnowledge, peers share choreographies, called interaction models, that specify how they have to interact in order to perform various distributed tasks.

[2] `http://www.openk.org`

The peers do not need to share an ontology, and a lot of effort is focussed on handling their heterogeneity. There are two situations in which matching between semantically heterogeneous elements are required: first, peers have to match the shared interaction models to their capabilities, and then, during the interaction, they have to translate the content of the received messages to their ontology. The work presented in this thesis tackles the second situation, improving the efficiency of peers and their ability to handle heterogeneity.

1.4 Thesis structure

Chapter 2 - Background In this chapter I present the background concepts relevant to this work: first I introduce the theory behind agent interactions and the formalism used to represent them, then I provide an introduction to ontologies, and a quick overview of ontology matching. At the end of the chapter, I briefly present the OpenKnowledge project, as an example of implementation of most of the ideas described in Chapter 2.

Chapter 3 - Assumptions and Motivations In this chapter I introduce the theoretical concepts: I describe and justify the assumptions that underpin the work, grounding them in the approach chosen for defining the interaction. The aims of the work (improving efficiency, recall, precision in ontology mapping and guiding the extension of ontologies) are also detailed.

Chapter 4 - Modelling context In this chapter, I first describe how the statistical model is built, interaction after interaction, and then how the model is used to predict the content of messages in new interactions. I also provide an example of the process of model creation and of computing the probability distribution for a message.

Chapter 5 - Evaluation In this chapter I present the evaluation of the system, discussing first the approach used for testing and then the results. The evaluation at first focuses on the ability of the predictor in providing a small set of suggestions that contains the correct correspondences with arbitrary probability, and then on the utility of the computed distribution in improving the performance of an ontology matcher. I also discuss how the utility of the predictor depends on the type of interactions: some interactions can benefit more than others.

Chapter 6 - Related work contains a more detailed literature overview of the relevant concepts introduced in Chapter 2. I first describe the different approaches to agent communication and service integration, and then I review some of the main approaches used in ontology matching.

Chapter 7 - Conclusion In this chapter I summarise the work, and present possible further work based on the current results.

Chapter 2

Background

2.1 Introduction

The goal of this work is to ease the communication between heterogeneous agents in open systems. The aim of this chapter is to introduce the main concepts in the domain and to show their grounding in the OpenKnowledge project. It is not a detailed overview: Chapter 6 is already dedicated to the literature review and presents the main stances of the research community on the topics introduced here.

Communication is about exchanging information, and requires the interacting actors to share a common set of signs and meanings. This work is concerned with the communication between software agents. What is meant with the term agent is presented in Section 2.3. Different approaches have been studied for multi-agents interactions: Section 2.4 in this chapter introduces the approach followed in this work, based on the concept of distributed workflows, which is at the basis of the OpenKnowledge project. A more in-depth overview of the various approaches is presented in Chapter 6.

Assuming that all agents share the same set of signs and meanings - a basic requirement for communication - has proved hard in open multi-agents systems. Ontologies, described in Section 2.5, are the formalisation of the meanings and signs used by agents. Heterogeneous agents may not share the same ontology: ontology mapping systems attempt to bridge different ontologies to allow interactions. Section 2.6 introduces the ideas and the problems related to ontology mapping. Section 6.4 in Chapter 6 provides a more detailed analysis of the different approaches in the literature.

Finally, Section 2.7 introduces the OpenKnowledge project, an implemented framework that deals with the issues presented in this chapter.

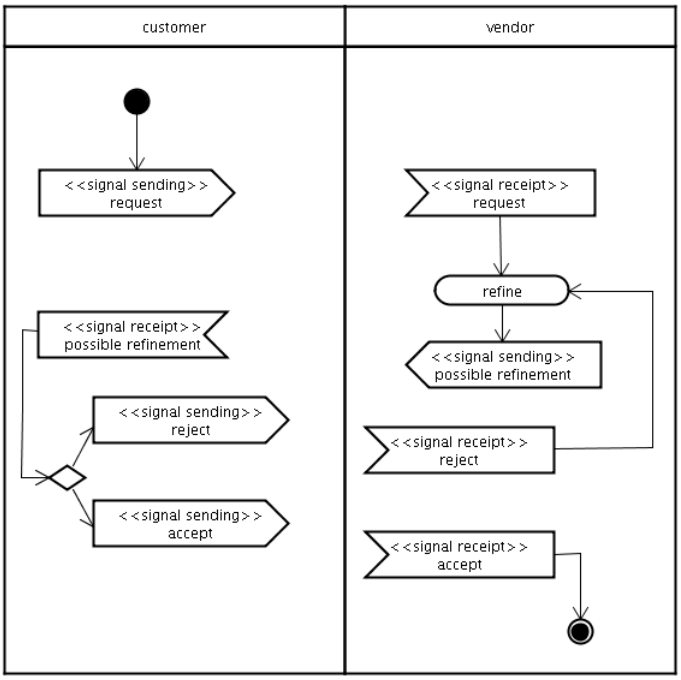

Figure 2.1: *Activity diagram for the scenario*

2.2 Example scenario

While the interaction framework used for this thesis allows to represent and run complex interactions involving any number of peers, a simpler kind of interaction is presented as an example scenario.

The scenario used is a subset of the classic *customer-vendor* scenario. At the start of such interactions, the customer asks the vendor for a product or service he would like to buy. However, as it is often the case, the customer may use a generic term, that can be interpreted in different ways. Therefore the vendor presents to the customer a selection of alternatives consistent with the request. The customer then chooses the option he prefers, and the interaction continues, for example to the payment, or to the definition of further details. The activity diagram in Figure 2.1 shows the flow of the messages between the customer and the vendor.

The interaction is generic and can be used in the purchase of different sorts of products or services, as Figures 2.7 and 2.8 show. However, the example followed throughout this thesis is relative to the booking of an accommodation for a conference, as shown in Figure 2.2. The participants are the customer and a travel agent: the customer starts by asking for a generic accommodation and the travel agent proposes different accommodation options, one of which is then selected by the customer.

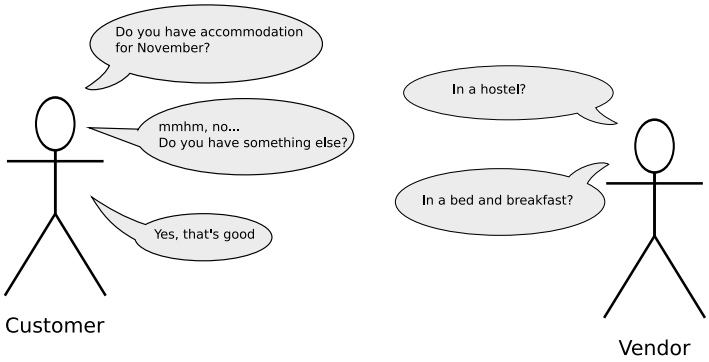

Figure 2.2: *Example scenario*

2.3 Agents and Peers

There is no shared and universally accepted definition of what an agent is. Wooldridge [67] defines an agent as:

> a computer system that is *situated* in some *environment*, and that is capable of *autonomous action* in this environment in order to meet its design objectives.

For the perspective used in this thesis, the autonomy of the agents involved in an interaction is not relevant, as the focus is on communication: in our example, the agent in the customer role may be a simple application used by a human user to contact the remote server of a travel agency, or a smart agent, instructed by a user to search for the best accommodation and entitled to spend real money. We use agent as synonym of actor or participant in an interaction.

In particular, as we have seen in the introduction and as we will explain more in detail in the next section, the interactions between agents are specified by choreographies that assign to all the participants the same relevance: therefore the term *peer* will also be used for the participants. In fact, in the OpenKnowledge project participants are peers in a peer-to-peer network.

2.4 Interactions

Many activities require interaction between different actors: in the example scenario, in order to book an accommodation an inquirer needs to contact a travel agency (or more than one) or directly a number of hotels.

In the simplest case, communication between two agents is a message transmitted from a sender to a receiver. According to speech act theory, a message is a performative

act that changes the state of the world [54]. The classical example used to explain this concept is the "*I do*" utterance pronounced in front of a registrar that causes the speaker to change his or her marital status. For example, a message sent from agents *i* to agent *j* to *inform* about ϕ will likely change the beliefs of *j*, adding the belief about ϕ. A more in-depth description of this "mentalistic" approach to communication can be found in Section 6.2. In our example, the following message, sent from the customer agent to the agent representing hotel Y:

```
inform(booking, 11 Nov 2008, 15 Nov 2008, Mr Smith, single)
```

should make the hotel agent believe that a single room must be reserved for the customer from the 11^{th} to the 15^{th} of November. Belief does not need to be conceived as the logical model described in the BDI architecture [11]: for belief we mean any internal representation of the information inside the agent. In this specific case of the example scenario, it can be a record in the database of the hotel system.

Usually interactions are more complex than single messages. The customer may first check the availability of offers, or it may want to first try single and then double rooms. Moreover, the booking may require a deposit or a credit card number. Or the hotel may inquire about other issues (breakfast, etc) related to the booking. This increased complexity, consisting in exchanges of messages, follows rules and conventions: as the conversation unfolds, the content of new messages is bound by the previously exchanged messages. A message failing to follow these rules would surprise the hearer as being off topic or even incomprehensible.

2.4.1 Dialogues and Interaction Models

Dialogues between software agents are, at least at the moment, simpler and more restricted than those between humans: they are carried out in order to reach a goal (buying a product, booking a flight, querying a price, etc.) and there is no need to care about digressions, unless relevant to the task. Therefore, their grammars can be simpler than those required for human interactions.

The rules and conventions that an interaction follows can be stated as sequences of messages hard-coded in the involved agents. They may be used to express preconditions and post-conditions for each each utterance: speech acts are considered actions and are combined into plans [11]. They can be defined in workflows that are followed as a script by the agents.

These approaches offer different trade-offs between flexibility and efficiency: embedding the interactions in the agents is the most inflexible but possibly very efficient. Planning offers the maximum flexibility but may require hefty computation at every interaction, and conditions can be difficult to verify. However, interactions are often repeated, so planning them every time is a waste of resources: workflows represent a good compromise and are currently the dominant solution.

2.4.2 Choreography and Orchestration

Workflows can either be conceived as centralised or distributed. In a centralised workflow, expressed through an orchestration language like BPEL [47] or YAWL [62], a single process executes the activities, and may call the other partners that are usually passive. In BPEL, calls are usually grounded to Web Service calls. In a distributed workflow, expressed through a choreography language like WS-CDL or LCC (see next section), the activities are executed by the various partners that communicate via messages.

In both approaches, a workflow describes an abstract set of activities and exchanged messages, not yet instantiated to particular values: it just defines where values come from, and where they go. For example, the workflow for booking a room starts expecting an input from the customer, who needs to specify dates, places, and preferences. The data are then forwarded to the hotel partner, that uses them as input for its local processing. The output of the processing, for example the request for further refinement, is sent back to the customer agent, who will use it as new input for further processing.

Workflows normally do not describe how the activities (like requesting input, or processing data) are performed: these are normally delegated via calls, either to the local agent or to a remote one via a web service. Agents answering to invocations can be set at design time, or can be found at execution time, exploiting some brokering mechanism. These calls may just verify a condition on some set of data, or introduce new data into the workflow: these calls are *sources* and introduce the problem of shared semantics of the data.

A source introduces terms according to its local semantics: these terms may then be used by the other partners in the interaction. This issue will be dealt in Section 2.5. Before proceeding to the problem of semantics, we first define the general requirements that a workflows language must satisfy in order to be used by the predictor and then

we describe the language that has been used in the implementation.

2.4.3 Workflow Language Features

A protocol can be modelled with a Finite State Machine (FSM) for each participant where the transitions consist of received messages or in the Boolean results of constraints (success or failure). The FSMs are defined by the entry-role for the participant peer and contain all the roles that the peer can take during an interaction.

During an interaction, the peer moves in the FSM, and creates a trace of the interaction. The variables in the trace are named and numbered to be unique. As interaction models can be recursive, the variables are tagged with their appearance in the run trace (in the example, the variable *Proposal* is used twice, so there will be two random variables named $Proposal_1$ and $Proposal_2$).

2.4.4 Lightweight Coordination Calculus

The *Lightweight Coordination Calculus (LCC)* [51, 52] is a choreography language based on π-calculus and can be used as a compact way of representing distributed workflows. Most workflow languages can be formalised using process calculi (such as π-calculus [48]). It is executable and it is adapted to peer-to-peer workflows. In the original version, interaction models are declarative scripts, circulated with messages. Agents execute the interaction models they receive by applying *rewrite rules* to expand the state and find the next move. Figure 2.3 defines the syntax of LCC. A full, formal description of a computation method for LCC is described in [53]. A summary of the rewrite rules is presented in Figure 2.4.

An interaction model in LCC is a set of clauses, each of which defines how a role in the interaction must be performed. Roles are described by their type and by an identifier for the individual peer undertaking that role. Participants in an interaction take their *entry-role* and follow the unfolding of the clause specified using a combinations of the sequence operator (‘*then*’) or choice operator (‘*or*’) to connect messages and changes of role. Messages are either outgoing to (‘$\Rightarrow$’) or incoming from (‘$\Leftarrow$’) another participant in a given role. A participant can take, during an interaction, more roles and can recursively take the same role (for example when processing a list). A message input/output or a change of role is controlled by constraints defined using the normal logical operators for conjunction and disjunction. There is no commitment to the method used to solve constraints, so different participants might operate different

$$
\begin{array}{rcl}
Model & := & \{Clause, \ldots\} \\
Clause & := & Role :: Def \\
Role & := & a(Type, Id) \\
Def & := & Role \mid Message \mid Def\ then\ Def \mid Def\ or\ Def \\
Message & := & M \Rightarrow Role \mid M \Rightarrow Role \leftarrow C \mid M \Leftarrow Role \mid C \leftarrow M \Leftarrow Role \\
C & := & Constant \mid P(Term, \ldots) \mid \neg C \mid C \wedge C \mid C \vee C \\
Type & := & Term \\
Id & := & Constant \mid Variable \\
M & := & Term \\
Term & := & Constant \mid Variable \mid P(Term, \ldots) \\
Constant & := & \text{lower case character sequence or number} \\
Variable & := & \text{upper case character sequence or number}
\end{array}
$$

Figure 2.3: *LCC syntax*

constraint solvers (including human intervention).

Figure 2.5 shows the initial part of an interaction model defining the interaction between a customer and a vendor described in Section 2.2: in this LCC fragment, the customer asks for a product and the supplier verifies if the request must be refined. If this is the case, the supplier will propose to the customer another, more specific, product. The customer, in turn, will analyse the proposal and see if it fits its needs. Interaction models are abstract descriptions of the interactions: they are then instantiated in real interactions. For example, the described interaction model can be used to specify the type of accommodation sought by a customer (Figure 2.7) or to specify the type of car a customer needs to rent (Figure 2.8).

A message in an interaction is a tuple, whose elements convey the content of a single communication act:

$m_i = \langle s_1, \ldots, s_n \rangle$

As we have seen above, a term s_i is introduced by some source: in LCC, constraints are sources. A terms is introduced by the agent solving the constraint via unification with its own knowledge base. In the example shown in Figure 2.7, *"accommodation"* is introduced by the customer, unifying the constraint `want(Product)` with its local knowledge to obtain `want(`*"accommodation"*`)`.

$R := B \xrightarrow{R_i M_i M_o \mathscr{S} \mathscr{O}} A :: E$	if	$B \xrightarrow{R M_i M_o \mathscr{S} O} E$
$A_1 \, or \, A_2 \xrightarrow{R_i M_i M_o \mathscr{S} O} E$	if	$\neg closed(A_2) \wedge A_1 \xrightarrow{R_i M_i M_o \mathscr{S} O} E$
$A_1 \, or \, A_2 \xrightarrow{R_i M_i M_o \mathscr{S} O} E$	if	$\neg closed(A_1) \wedge A_2 \xrightarrow{R_i M_i M_o \mathscr{S} O} E$
$A_1 \, then \, A_2 \xrightarrow{R_i M_i M_o \mathscr{S} O} E \, then \, A_2$	if	$A_1 \xrightarrow{R_i M_i M_o \mathscr{S} O} E$
$A_1 \, then \, A_2 \xrightarrow{R_i M_i M_o \mathscr{S} O} A_1 \, then \, E$	if	$closed(A_1) \wedge A_2 \xrightarrow{R_i M_i M_o \mathscr{S} O} E$
$C \leftarrow M \Leftarrow A \xrightarrow{R_i M_i M_i - \{m(R_i, M \Leftarrow A)\} \mathscr{S} \emptyset} c(M \Leftarrow A)$	if	$m(R_i, M \Leftarrow A) \in M_i \wedge satisfy(C)$
$M \Rightarrow A \leftarrow C \xrightarrow{R_i M_i M_o \mathscr{S} \{m(R_i, M \Leftarrow A)\}} c(M \Rightarrow A)$	if	$satisfied(\mathscr{S}, C)$
$null \leftarrow C \xrightarrow{R_i M_i M_o \mathscr{S} \emptyset} c(null)$	if	$satisfied(\mathscr{S}, C)$
$a(R,I) \leftarrow C \xrightarrow{R_i M_i M_o \mathscr{S} \emptyset} a(R,I) :: B$	if	$clause(\mathscr{S}, a(R,I) :: B) \wedge satisfied(\mathscr{S}, C)$

An interaction model term is decided to be closed as follows:

$$\begin{aligned} &closed(c(X)) \\ &closed(A \, then \, B) \leftarrow closed(A) \wedge closed(B) \\ &closed(X :: D) \leftarrow closed(D) \end{aligned} \tag{2.1}$$

$satisfied(\mathscr{S}, C)$ is true if constraint C is satisfiable given the peer's current state of knowledge.

$clause(\mathscr{S}, X)$ is true if clause X appears in the interaction model $\mathscr{S}$, as defined in Figure 2.3.

Figure 2.4: *Rewrite rules for expansion of an interaction model clause*

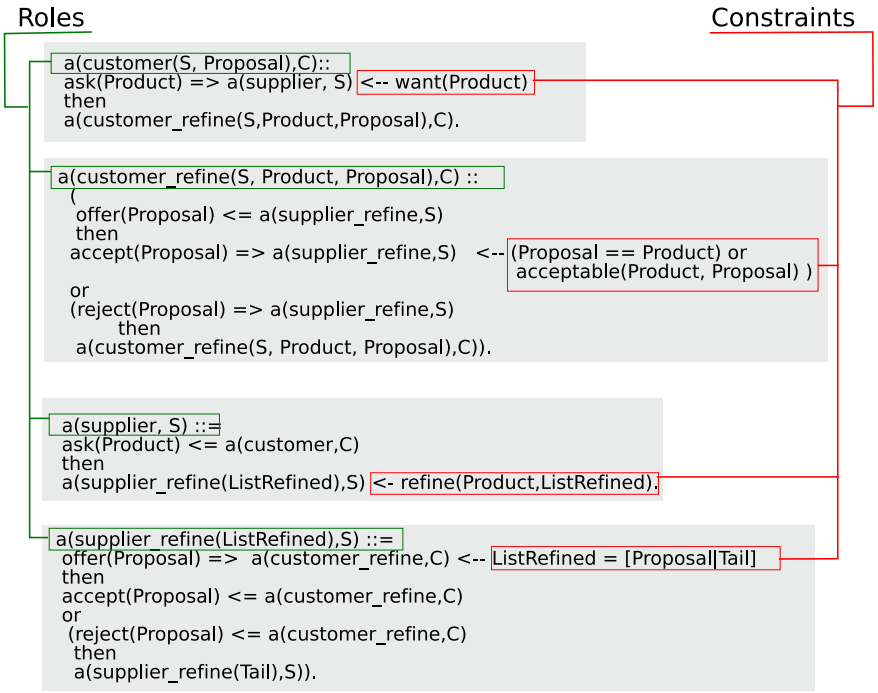

Figure 2.5: *Request refinement in LCC*

Previous work on LCC includes the generation at run-time of interaction models [38], the creation of successful teams for interactions [35], the distributed relaxation of constraints [32] and the formal verification of properties of the interaction models [45].

LCC has been used in applications such as business process enactment [30] and e-science service integration [3]. In particular, it has been chosen as the specification language used for defining interaction models in OpenKnowledge, as we will see more in detail in Section 2.7.

Compared to other languages like BPEL or YAWL, LCC is surely more compact, even though it does not allow the same level of specifications. Some of these limitations have been overcome in OpenKnowledge, extending LCC with annotations. Any element in an interaction model can be annotated: it is possible, for example, to annotate a variable in a role, specifying its semantic type. However, the main difference with the other orchestration languages is that it is possible to express the behaviour of all the participants. A YAWL or BPEL workflow defines the behaviour and keeps the state of only one participant: the other are just passive components that are invoked, and are unaware of being involved in a run of a workflow. More complex interactions, such as auctions where behaviour of all participants should be defined, are thus more difficult to represent in languages YAWL or BPEL, based on a centralised paradigm.

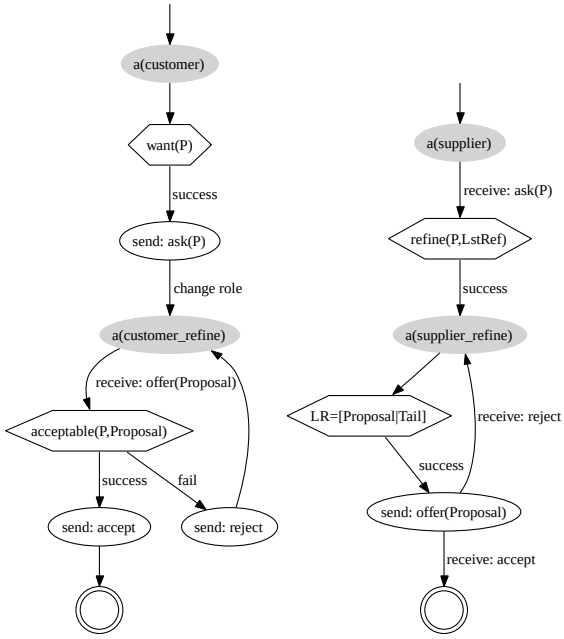

Figure 2.6: *Finite State Machine for the entry role* customer *and* supplier

2.4.5 Matchmaking

Constraints in LCC, or service invocations in other workflow languages, are performed by some agent, that must be identified at some stage of the process.

In many orchestration-based languages like BPEL the participants are defined at design time. In more flexible systems, agents and interactions can be composed at run-time. Flexibility is reached through search: given an interaction, agents can be found or, given a group of agents, an interaction can be selected. Adaptors are often required in open systems, where agents and interactions do not share the same representation. For example, languages like BPEL or YAWL provide a set of operations (based on XPath and XSLT) for transforming the data before invoking services which use different formats.

In simple client/server architectures, a client will search for an appropriate server in order to perform a task (like booking a room). The query will return the possible servers, each with its specific interaction model that the client will follow. Other architectures, such as OpenKnowledge, decouple the interaction models from the participants: an agent may first look for an interaction fitting its needs, and then search for other participants willing to take part in it.

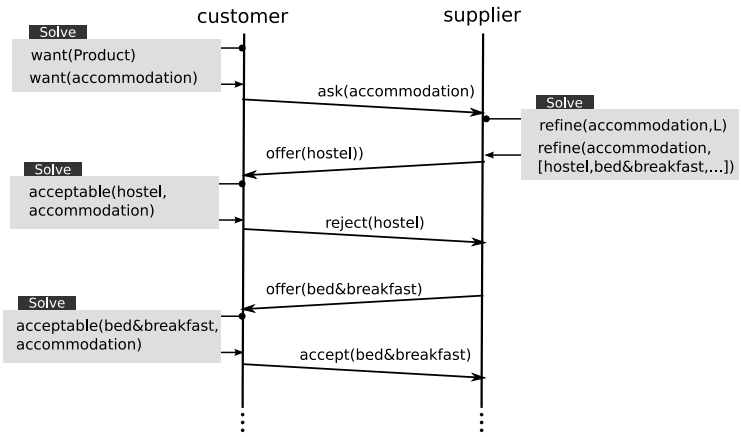

Figure 2.7: *Run of the interaction model in Figure 2.5 for refining an accommodation request*

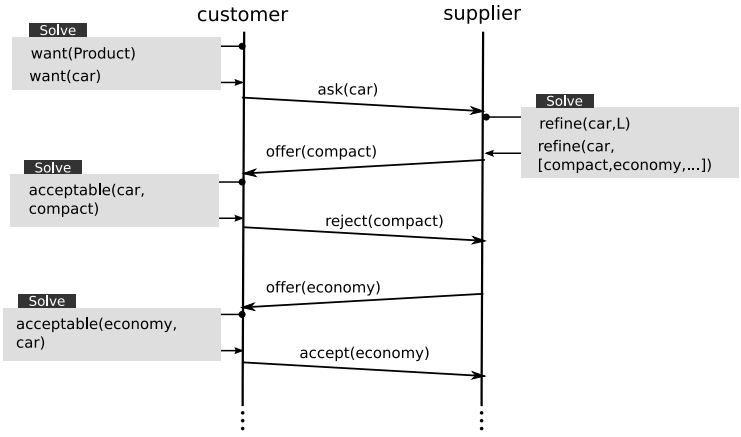

Figure 2.8: *Runs of the interaction model in Figure 2.5 for refining a car rental request*

2.5 Ontologies

Interactions participants have their knowledge and skills: they provide points of access to information repositories, they provide services that process information and so on. *Ontologies* are used to name and define the elements in the knowledge bases. The term *ontology* (from the Greek words meaning *being* and *science, study, theory*) comes originally from philosophy, where it means the study of what exists, and forms the main subject of metaphysics. In Artificial Intelligence what exists is what can be represented. According to Gruber:

> An ontology is an explicit specification of a conceptualization.[27]

This definition was then extended to include the idea that the conceptualisation should be shared among different parties:

> An ontology is a formal, explicit specification of a shared conceptualization. [59]

Ontologies are often compared to database schemas, with which they share some similarities: they both provide a vocabulary of terms that describe a domain of interest and constrain the meaning of the terms used in the it. However, a database schema does not provide an explicit semantics for their data, while ontologies are logical systems, that obey to some formal semantics: we can interpret the ontological definitions as a set of logical axioms [43]. Ontologies are often distinguished by their level of generality:

- **Domain ontologies**: they capture the knowledge of a specific domain. Examples of domain ontologies are:
  - the Engineering Mathematics ontology [28],
  - the Enterprise Ontology [61] and the TOVE ontology [29] for representing business models,
  - the Software Engineering Body of Knowledge (SWEBOK[1]) [8],
  - the Unified Medical Language System (UMLS[2]) [60],
  - the GeneOntology, providing a "*controlled vocabulary to describe gene and gene product attributes in any organism*"[3],
  - the United Nations Standard Products and Services Code (UNSPSC[4])
- **Upper ontologies**: they attempt to describe general concepts valid across all domains. Examples of upper ontologies are:
  - Cyc[5] [36],
  - the Suggested Upper Merged Ontology (SUMO[6]) [41],
  - the Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE[7]) [42],

[1] http://www.swebok.org/
[2] http://www.nlm.nih.gov/research/umls/
[3] http://www.geneontology.org/
[4] http://www.unspsc.org/
[5] http://www.cyc.com/
[6] http://www.ontologyportal.org/
[7] http://www.loa-cnr.it/DOLCE.html

– the Basic Formal Ontology (BFO[8]) [26]

The term of ontology is often used to refer to taxonomies (simple hierarchies of terms): for example GOOGLE uses the DMOZ[9] ontology, result of a collaborative effort, to categorise websites, while Amazon and eBay use ontologies to classify their products.

2.5.1 Ontology formalisation

According to [65], an ontology is composed by definitions of classes, relations or instances. The definitions of these entities are tuples:

$Def = \langle T, D, C \rangle$

where T is the term that identifies the entity to define (*definiendum*, meaning "thing to be defined" in Latin) and it is an atomic formula in a formal language; D is the formal definition (*definiens*, meaning "defining thing") and it is a possibly compound formula in a formal language; C is the concept description, obtained in the conceptualisation step, and can be expressed in natural language.

The predictor presented in this thesis can use taxonomies with properties: if the ontology is a simple taxonomy of classes, the definition D is the hierarchy of the classes subsuming the entity to define. The concept description C can either be explicitly written in the ontology (for example using the tag `rdfs:comment` in a rdf/owl ontology), or can be an implicit meaning conventionally associated to the term, and normally recognised in a dictionary.

Figures 2.9 and 2.10 show a portion of the customer and vendor ontologies in the example scenario. According to the definition above, a term like *"restaurant"* in customer's ontology can be defined as:

T : *restaurant*
D : $restaurant \equiv (has\_cuisine.cuisine) \sqsubseteq eatery \sqsubseteq thing$
C : *"a building where people go to eat."*[10]

Different formal languages have been developed to represent ontologies, at different levels of expressivity (and computability): from KIF [19], developed in the 90s, based on first order logic and aimed at knowledge sharing, to the OWL family [58], based on different variants of Description Logics[2] ($\mathcal{SHOIN}(\mathcal{D})$ for OWL-DL,

[8] `http://www.ifomis.uni-saarland.de/bfo/`
[9] `http://www.dmoz.org`
[10] according to WORDNET 2.0

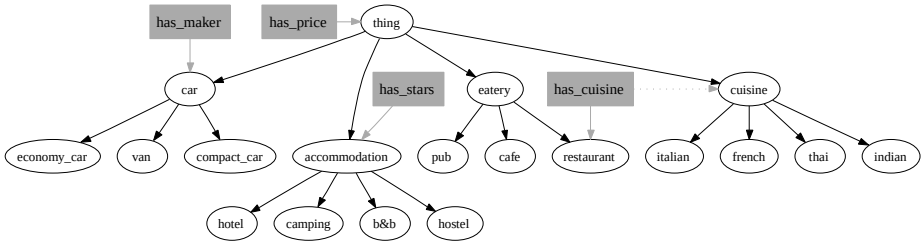

Figure 2.9: *Customer ontology. Circles are concepts, grayed boxes are properties*

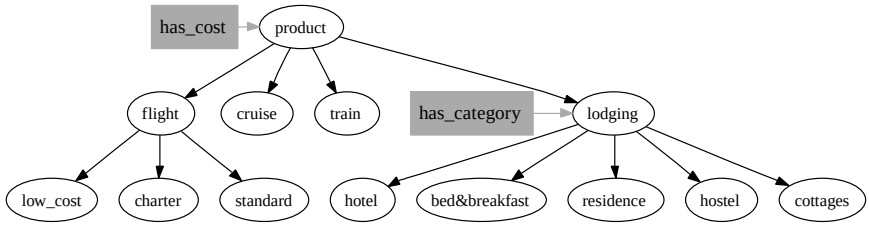

Figure 2.10: *Vendor ontology. Circles are concepts, grayed boxes are properties*

the less expressive $\mathscr{SHIF}(\mathscr{D})$ for OWL-lite), and oriented towards the Semantic Web.

2.5.2 Problems of shared ontology

Ideally, a common, shared ontology should have appeared, allowing complete interoperability between the agents. But imposing the same ontology on all agents has proved difficult and impractical. Firstly some "social" problems arise. There is often a choice of different ontologies for a specific purpose: for example, we saw earlier that Cyc, SUMO, BFO or DOLCE are alternative upper ontologies. Who imposes which ontology should be used? Why should the others accept it? Even in case one ontology is finally chosen, many “legacy” ontologies keep being used [31].

It is also difficult to keep track of the evolution of an ontology: some agents may keep the pace with the updates, while others may remain with out of date versions. As described in [31], different versions of the same ontology can sometimes be treated as different ontologies. In general, differences in the interests and needs can make it difficult to create a consistent ontology that takes into account all the views.

As a clear indication of the number of developed ontologies, the entry page of

SWOOGLE[11] [12], a search engine for ontologies, states "*Searching over 10,000 ontologies*". Searching on the engine the synonym "*lodging*" yields 12 different ontologies, the term "*hotel*" yields 62, and the term *"car"* yields more than 250.

2.5.3 Sources of ontological heterogeneity

Ontologies can differ for various reasons: Section 6.4.1 presents a classification of the mismatches categorisations in literature. In brief, the mismatches can rise because:

- *the same name or formal definition is given to different concepts:* $(T_1 \equiv T_2 \vee D_1 \equiv D_2) \wedge C_1 \neq C_2$. For example, the term *bank* can mean a slope, an array of elements, or a financial institution, or a flight manoeuvre
- *a different name or formal definition is given to the same concept:* $(T_1 \neq T_2 \vee D_1 \neq D_2) \wedge C_1 \equiv C_2$. In the two example ontologies, *accommodation* and *lodging* mean the same concept, even though their name is different and their formal definition is different (their superclasses are different and the properties have a different name)

2.6 Ontology matching

The emergence of different ontologies, and the problem of agreeing on a shared one have pushed researchers to study methods for bridging them. The various attempts to reconcile ontologies can be divided into *merging*, *aligning* and *integrating* [31]. Merging is the act of building a new ontology by unifying several ontologies into a single one, typically when two big companies merge and need to unify their knowledge bases; matching is used when sources must be made coherent and consistent, but must be kept separated; finally, integrating entails building a new ontology composing parts of other ontologies. However, matching ontologies lies at the basis for both merging and integration.

Ontology and schema matching are used in many fields. Traditional approaches include catalogue integration for e-business, distributed query processing, data warehousing. These applications are based on design time matching operation. Catalogue integration, for example, requires to identify the correspondences between entries, in

[11] `http://www.swoogle.org`

order to generate queries that translate data instances in the catalogues, providing a unified access point to the data [56].

In recent years, a new emerging set of applications, characterised by dynamicity, has been added. In these applications, ontology alignment is often performed at run-time and used to provide interoperability between heterogeneous peers in P2P systems, allow agents to understand speech acts specified in different ontologies [63], or allow dynamic web service integration [46].

2.6.1 Ontology matching definition

An ontology matching algorithm is a function that receives two ontologies O_1 and O_2, some auxiliary resources R (such as a thesaurus) and returns the alignment C between their entities:

$$match : O_1 \times O_2 \times R \rightarrow C \tag{2.2}$$

where the alignment C contains all correspondences between entities in O_1 and O_2. The correspondence for a term $w_i \in O_1$ is normally found by comparing it with a list of terms $T \subseteq O_2$:

$$findCorrespondence : w_i \times T \times O_1 \times O_2 \times R \rightarrow \rho \tag{2.3}$$

where ρ is the correspondence and it is defined by the best relation r_k found (among the possible ones, such as similarity, equivalence, subsumption,etc), with confidence c, between the term $w_i \in O_1$ and another $t_j \in T$ (where normally $T \equiv O_2$):

$$\rho = \langle id, r_k, w_i, t_j, c \rangle$$

The problem is how to verify the existence of a particular relation $r_k(w_i, t_j)$ between the terms w_i and t_j from two different ontologies. If the ontologies are mutually inconsistent, as it is often the case, it may be impossible to prove the relations using logic reasoning from the definitions in the ontologies or, even worse, wrong relations may be derived. Therefore, matching algorithms need to use other methods to identify relations between entities in different ontologies. These methods usually assume that ontologies share some identifiable similarities. For example, the similarities can be in the label used to identify the entities, in their formal definition, or in the description (possibly implicit) of the concepts attached to the entities.

The task is made more difficult by the vagueness or ambiguity of the terms (for instance, the terms may have many different senses, with only a few overlapping) and by the lack or the imprecision of the information available in the process (for example a term or a sense may not be included in a thesaurus).

The value t_j in the result of the *findCorrespondence* function can be modelled as a *random variable*. A priori, before applying the matcher, all the relations between $w_i \in O_1$ and all terms $t_j \in T \subseteq O_2$ are equally probable:

$$P\left(r_k\left(w_i, t_j\right)\right) = P\left(r_h\left(w_i, t_g\right)\right) \;\; for\, w_i \in O_1 \; and \; \forall t_j, t_g \in T \subseteq O_2 \tag{2.4}$$

The function *findCorrespondence* uses the result of *matchers* that extract information about the similarities between terms w_i and t_j: the various techniques used in the literature are reviewed in Section 6.4 in Chapter 6. The operation of collecting information can be qualitatively modelled as gaining evidence in order to obtain an approximate posterior probability distribution of the relations between w_i and the terms in the other ontology:

$$\mathbf{P}(w_i \times R \times T \,|\, matchers\, results)$$

where the domain $w_i \times R \times T$ is the product between the foreign term w_i, the possible relations R and the selected list of terms T to compare and *matcher results* is the list of all the results of the comparisons between w_i and the terms in T. Assigning these posterior probabilities is difficult, and often arbitrary. For example, a matcher using only string comparison may have obtained an edit distance[12] of 1 between w_i and t_k and t_j, and equal or higher than 2 between w_i and the remaining terms. Without any additional information, the probability that w_i is mapped by t_k or t_j is arbitrary, and could be set - for example - to 50% each, excluding that terms with higher distances are the correct correspondence. Some matching algorithms work iteratively, using more certain information collected in previous iterations to increase the available information: for example, if the term t_k was already mapped with high probability to another term w_p in O_1, then it is possible to add this information in the evidence available to $r_h(w_i, t_k)$; similarly if the neighbours of the term w_i are already mapped to the neighbours of t_k but not to those of t_j, then it is possible to increase the information available for $r_h(w_i, t_k)$.

[12] number of alterations needed to transform one string into the other

2.6.1.1 Evaluating the matching systems

The quality of a matching system is usually measured by its *precision* and its *recall*, or their aggregation represented by *F-Measure*. Given that M_{found} is the set of correspondences found by the mapping system, $M_{correct}$ is the set of correct correspondences, usually defined by human experts:

Precision is the ratio between the number of correct correspondences among those found and the total number of found ones:
$Precision = \frac{|M_{found} \cap M_{correct}|}{|M_{found}|}$

Recall is the ratio between the number of found correspondences and the total number of possible ones:
$Recall = \frac{|M_{found} \cap M_{correct}|}{|M_{correct}|}$

F-measure is the harmonic mean of recall and precision:
$F-measure = \frac{2 \times Prec \times Recall}{Prec + Recall}$

While in toy ontologies most of the systems work well and obtain high precision and recall, in real world ontologies the recall is fairly low, as shown in [17]. This is because the matchers often lack the background - or domain specific - knowledge needed to extract the similarities between two terms, and therefore they cannot influence the probability distribution of the relations, making it impossible for the decision process to select the best correspondence.

2.7 OpenKnowledge

The ideas presented in this chapter find a grounding in the EU funded OpenKnowledge[13] project, that involves the universities of Edinburgh, Trento, Amsterdam, Barcelona and the Knowledge Media institute (KMi) in the Open University. The aim of the project is to create an architecture for an open, coordinated knowledge sharing system, which anyone can join at any time: the result of this project is an executable peer-to-peer framework[14], in which peers interact using shared interaction models. I was involved as software developer for the OpenKnowledge kernel, and during the implementation of the framework we encountered many of the issues previously discussed:

[13] http://www.openk.org
http://cordis.europa.eu/ist/kct/fp6_openknowledge.htm
[14] http://www.cisa.informatics.ed.ac.uk/OK/download/ok.zip

the engineering decisions taken to solve them represent an interesting comparison and can help their understanding.

The core concept in OpenKnowledge are the interactions between participants, defined by *interaction models* written in LCC and published by the authors on the *distributed discovery service* with a keyword-based description. The roles in the interaction models are played by the participants, called *peers*. The peers that want to perform some task, such as booking a room or providing a booking service, search for published interaction models for the task, and then advertise their intention of interpreting one of its roles to the discovery service for the specific task by subscribing to it. In the scenario relative to the interaction shown in Figure 2.7, a travel agency P_1 has subscribed to perform the role of `supplier` for a task *"room booking"*, while a peer P_2 searching a room has subscribed as `customer`, for a task described similarly (for example, just *"room"*). For the interaction in Figure 2.8, a car rental agency P_3 has subscribed to perform the role of `supplier` for a task described as *"car rental, car hire"*, and the peer P_4 looking for a car has subscribed as `customer`, for a task defined as *"car rental"*.

When all the roles are filled, the discovery service matches the peers which subscribed for the same or similar tasks (for example, peers P_1 and P_2 with their descriptions *"room booking"* and *"room"* or peers P_3 and P_4 with their descriptions *"car rental, car hire"* and *"car rental"*), and then chooses randomly a peer in the network as coordinator for the interaction, and hands over the interaction model together with the list of involved peers in order to execute it.

The coordinator first asks each peer to select the peers they want to interact with (a customer may want to buy from a specific vendor, and not from any vendor), composing a mutually compatible group of peers out of the replies, and then asks the peers to commit. If the peers commit, then the coordinator can execute the interaction, instantiating a local proxy for each peer. The remote peers are contacted only to solve constraints in the role they have subscribed. In the example interaction model, the coordinator will ask the peer that has subscribed as `customer` to solve `want(Product)`.

Figure 2.11 shows the lifecycle of the OpenKnowledge framework, from the selection of an interaction to its execution.

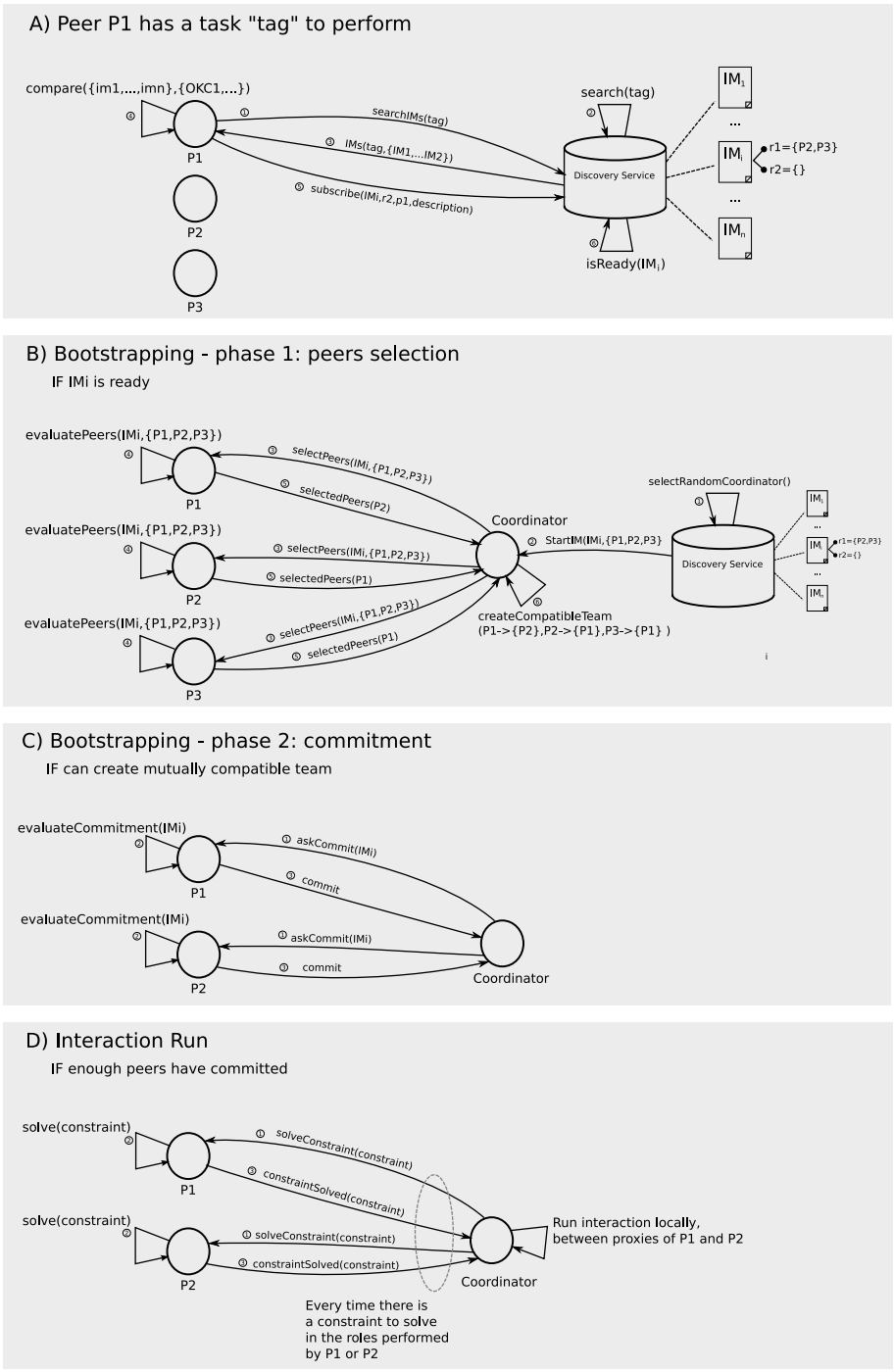

Figure 2.11: *OpenKnowledge lifecycle*

2.7.1 What is a peer in OpenKnowledge

A peer is simply a node in a peer-to-peer network. It can be a GUI-based application, directly used by a human user, or a server application. The peer-to-peer network is accessed via the *ok-kernel*, that provides the basic functionalities for sharing, searching, subscribing to or taking part in interactions.

Peers involved in an interaction are contacted by the coordinator in order to solve constraints: to this end, they use the methods provided by their locally installed components.

2.7.2 Matchmaking in OpenKnowledge

Selecting the interaction

A peer interested in performing a task queries the discovery service for a published interaction model matching a provided description. The discovery service returns the list of all the models whose description is similar to the given one.

The peer compares the list of received interactions with the methods it has in its local components, ranking the interactions based on its capabilities to perform them. The ranking of the interactions can be influenced also by their popularity (how often they have been used), a measure given by the discovery service.

Selecting the peers

The peers proactively search interactions and actively subscribe to them: peers subscribed to an interaction are peers interested in taking part in them. However, a peer may not accept all combinations of peers: for example a buyer may want to buy a product only from vendor A, but not from vendor B, even though they are both subscribed as sellers to the same interaction.

Therefore, before taking part in an interaction, all the peers subscribed to it are asked by the coordinator to select who they want to interact with. Peers can have internal models to represent the reliability of other peers, depending on their previous experience with them, and can share these information with others or use the ratings already collected by others.

2.7.3 Ontology matching in OpenKnowledge

One of the founding motivation of OpenKnowledge is the openness of the system: as we saw, any peer can join at any time, subscribing to a particular interaction. Because

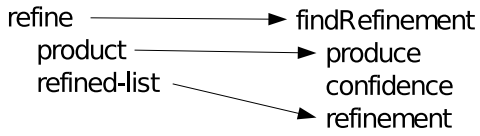

Figure 2.12: *Example of structure matching between the constraint* `refine(product, refined-list)` *and method* `findRefinement(produce, confidence, refinement)` *in an OpenKnowledge component.*

of this openness, peers can be widely heterogeneous, and therefore the alignment between different ontologies used by peers plays a fundamental role.

There are two types of matchings that a peer needs to perform in order to participate meaningfully to an interaction: one offline (at subscription time) and one online (during the interaction).

Offline matching

Matchmaking requires offline matching:

- the discovery service needs to expand queries to match them against the stored descriptions of published interaction models
- the peers need to compare the constraints in the received interaction models with the methods in their local components

The parameters in the constraints are annotated with their semantic types. Similarly, parameters in the methods of the local components are marked up with terms from an ontology, possibly different from the one used in the interaction annotations. When a peer needs to perform a task, asks the discovery service for a list of interaction models, and matches them with its own components using tree matching [25, 18]. The result of the matchings provides a measure of the distance between the interaction model and the peer capabilities [22], together with the set of adaptors between the constraints and the methods in the peer's components. The peer selects the interaction model that fits best, and then uses the computed adaptors. An example of adaptor, used to match the constraint `refine(Product, RefinedList)` in the scenario interaction model to a method in a plug-in component, is shown in Figure 2.12.

Online matching

When a peer subscribes to an interaction often it cannot know which other peers will

subscribe to the interaction: in the example interaction, the supplier subscribes first, and then wait for the other peers to subscribe as customers. Only when the interaction starts the peers will be given the list of all the peers and will select those who they are willing to interact with.

Even at this point they do not know yet who they will actually interact with, because the coordinator use the preferences of all the subscribed peers in order to make a mutually compatible group of peers. Therefore it makes sense for the peer to wait until it receives the constraints with the foreign terms and map them at run-time. The approach presented in this thesis aims at tackling this problem: Chapters 3 and 4 discuss it in detail.

2.8 Summary

This chapter has introduced the main concepts needed as background knowledge for understanding the research presented in this thesis: facilitating the interaction among heterogeneous agents.

We have seen that, while an agent is usually intended as an autonomous actor, in this work the term agent simply means participant in an interaction. We have also seen that while interactions can be planned dynamically, often agents only need to repeat over and over the same type of interactions: executable workflows can be used as an efficient and clean compromise, and it has been chosen as solution in this work. Agent can execute different workflows depending on their objective. The interactions are described in LCC, a declarative, executable language based on π-calculus: a LCC script defines the distributed workflow the various agents must execute.

Agents have ontologies, which formally define the terms they can use in reasoning about their domain. The agents involved in the interactions may not share the same ontologies, and therefore communication implies creating bridges between the ontologies using some of the available ontology matching algorithms.

The OpenKnowledge project offers a running framework implementing the ideas presented in this chapter: it is a peer-to-peer system where peers interacts through shared interaction models written in LCC.

Chapter 3

Conceptual Framework

3.1 Introduction

We have seen in Chapter 2 that the most basic interaction is a single message, that changes the internal state of the recipient. This assumes that all agents in the interactions are able to understand the messages, because they share the ontology defining the possible terms. But this may not be the case: we have seen that agents may have different ontologies, and therefore they need to have access to the correspondences between them.

As we have introduced in Section 2.6 and will discuss in Section 6.4, many different ontology mapping systems have been developed and tested, . The core problem encountered by the mapping systems is that they aim at a full ontology commitment between the agents: they try to find an agreement on the meaning of as many terms in the ontologies as possible. As we have seen, this has proved harder than expected. In

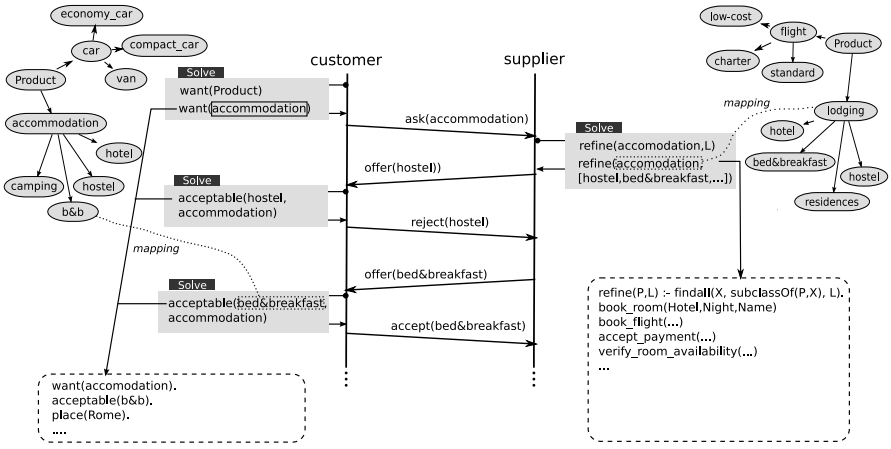

Figure 3.1: *Applying matching in an interaction*

an open system like OpenKnowledge it is infeasible to precompute all the correspondences offline, as it impossible to know in advance all the participants in an interaction: correspondences must be computed dynamically when interactions take place. For example, as we have seen in Section 2.7, the supplier peer in the interaction shown in Figure 3.1 cannot know the customer's identity until the interaction starts.

If the peers perform every time different tasks, using different interaction models, there would be little useful information that could be extracted by observing the interaction runs. However, when the peers need to perform the same task, they will likely use the same interaction model, and will probably exchange similar messages. This repetition can be exploited to learn and build a model of the content of the interaction. As we make clear in Section 3.4, the assumption is that there are relations between terms in different messages, and that terms appear with different frequencies. Terms have relations because dialogues are constrainted by rules and conventions, made explicit by the use of interaction models. Terms in a message may have different frequencies because of three main reasons: first, some of the terms may be unrelated to the interaction model, and therefore will appear rarely, second, their frequencies may reflect the needs and desires of the community that uses the interaction model in a certain period of time, third, their use depends on the specific context of an interaction run.

The model obtained analysing the content of various runs of an interaction model can be used to predict the content of future interactions. The prediction is a probability distribution of the terms in a particular transition of an interaction, such as a received message, given the current state and the history of the previous runs of the interaction. As we will see in Section 3.5, the prediction can be used for improving the efficiency of the ontology mapping oracle, suggesting a subset of most likely terms to verify. It can be used as additional evidence to the information collected by the mapping oracle in order to improve its precision and recall. It can also be used as a source of suggestions for extending the ontology.

3.2 Problem definition

The agents execute the interaction model inside a separate "box". The "box" in which an interaction model is run can be compared to the idea of *context* described by Ganglia: in [20] he defines a context c_i as "*partial*" and "*approximate*" theory of the world, represented by the triplet $\langle O_i, A_i, \Delta_i \rangle$. In the tuple, O_i is the language local to the con-

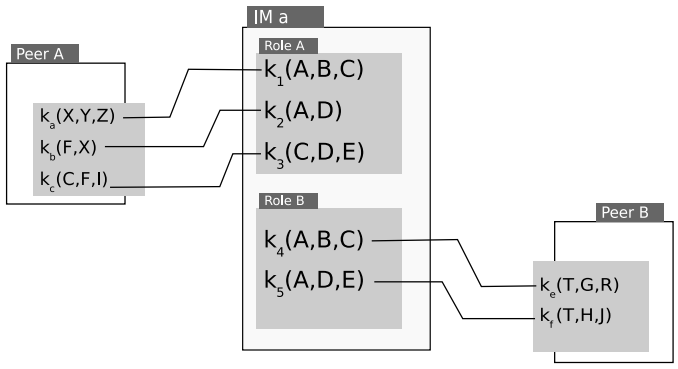

Figure 3.2: *Bridges between the environments*

text, A_i is the set of axioms of the context, and Δ_i is the inference engine local to the context. Moreover, a reasoner can connect a deduction in one context with a deduction in another using *bridge rules*.

For the context of an interaction model run $c_r = \langle O_r, A_r, \Delta_r \rangle$, the language O_r is composed by all the terms that can be introduced by the agents involved in the interaction; the axioms A_r are the role clauses and Δ_r is the interaction model expansion engine (see Section 2.4.4).

Interaction models can be executed if it is possible to bridge the reasoning between the interaction context c_r and the agent's local context c_a. This is accomplished finding the bridge rules that connect the constraints in the interaction model with the predicates in the agent's local knowledge:

$$\frac{c_r : \kappa_p(W_1, ..., W_n)}{c_a : \kappa_a(T_1, ..., T_m)} \; where W_i \in O_r, T_j \in O_a \tag{3.1}$$

where κ_p is a formula of an interaction model constraint and κ_a is a formula in the agent's local knowledge, that can be satisfied only by using its own language O_a, which is the peer's ontology.

In traditional ontology mapping, the bridges should be valid for any value from L_r and L_a in two contexts c_r and c_a:

$$\forall W_1 ... W_n \in L_r, \exists Y_1 ... Y_n \in L_a.\ c_r : \kappa_p(W_1, ..., W_n) \rightarrow c_a : \kappa_q(T_1, ..., T_m) \tag{3.2}$$

or alternatively:

$\forall W_i \in O_r, \exists T_j \in O_a.\ ref(W_i) \simeq ref(T_j) \simeq Q_k$

That is, for any value of $W_1, ..., W_n$ in κ_p, it is possible to find the values for $T_1, ..., T_n$ so that $c_a : \kappa_q$ is equivalent to $c_r : \kappa_p$. In the example scenario, the correspondences

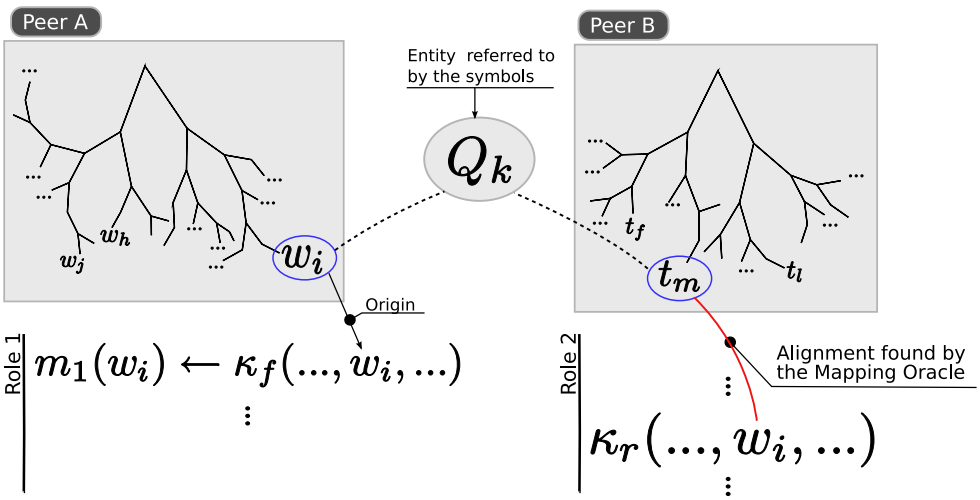

Figure 3.3: *Translation problem: a term w_i inserted by peer A needs to be used in a constraint by peer B. The term w_i refers to some unknown entity q_k: the matching term t_m must refer to the same entity for the communication to be meaningful.*

should cover the possible requests from the customer agent for buying any element in its ontology even if these interactions never take place.

This is a strong requirement: it assumes that it is possible to find a corresponding term in O_a for every term in O_r, and this may not always be the case. It is possible to limit the correspondences to those needed to perform the occurring interactions, and with no need to guarantee complete equivalence between the languages. Therefore an agent needs to map only the terms that appear in $c_r : \kappa_p$ in order to satisfy $c_a : \kappa_q$:

$$\exists W_1 \ldots W \in O_r, T_1 \ldots T_n \in O_a.\ c_r : \kappa_p(W_1, \ldots, W_n) \wedge c_a : \kappa_a(T_1, \ldots, T_m) \tag{3.3}$$

that is a much weaker requirement: we need to find the values for $T_1, \ldots, T_n$ so that $c_a : \kappa_a$ is valid for the given instances of $W_1, \ldots, W_n$. In the example, it means that only the correspondences required for booking the room are needed.

Let us suppose that a peer, with ontology O_a, needs to satisfy a constraint $\kappa_r(\ldots, w_i, \ldots)$ when in a specific state of an interaction, and that $w_i \notin O_a$ is the foreign term. The task is to find what entity q_k, represented in the agent's ontology by the term $t_m \in O_a$, was encoded in w_i. The term t_m is the matching term: it is, in the agent's ontology, the closest to the intended entity q_k. For our work, the matching term is assumed to exist in O_a.

The matching is performed by a "*mapping oracle*", whose specific implementation

is irrelevant for this work: any existing mapping system, such as S-Match [23], would fit smoothly in the framework.

In the example scenario of Figure 3.1, in order to satisfy the constraint `refine (Product, List Refined)`, the supplier must map the term "*accommodation*" to *"lodging"* in its ontology.

3.3 Predicting the content of messages

The intended entity q_k represented by the foreign term w_i is, from the agent's perspective, an *event* of a random variable Q_k, whose domain is the whole ontology. As said before, an ontology mapping algorithm can be used to interpret the sign w_i in the message and finds the corresponding symbol t_m.

However, conventional ontology mapping algorithms do not take into account the context of the interaction , and consider, before applying the matchers, all the terms in the domain as improbable:

$$P(Q_k = t_i) = P(Q_k = t_j) \; for \; \forall t_i, t_j \in O_a$$

As introduced earlier, dialogues follow conventions and rules, made explicit by the interaction model, and the content of the messages are influenced by the local and the general context: therefore the terms are not improbable - some will be more likely than others.

Our main claim is that the random variable Q_k has a conditional probability distribution, similar to the one in Figure 3.4, where the evidence is the context of the interaction:

$$\mathbf{P}\left(Q_k \left| IM_{state}, IM_{history}\right.\right) = \left\langle \begin{array}{c} P(Q_k{=}t_1 | M_{state}, M) \\ \vdots \\ P(Q_k{=}t_n | IM_{state}, M) \end{array} \right\rangle \tag{3.4}$$

where $t_1 \ldots t_n$ belong to the peer's ontology and $p\left(Q_k = t_i \left| IM_{state}, M\right.\right)$ is the probability that t_i is the best matching term for Q_k, given the statistical model M of the interaction, obtained from previous runs of the interaction model, and the current state of the interaction. The current state IM_{state} of the interaction is given by the values of all the variable substitutions up to the message currently processed:

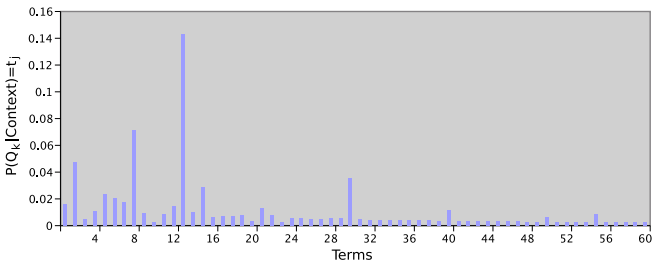

Figure 3.4: *Example of probability distribution for a variable Q_k*

$$IM_{state} = \left\{ \begin{array}{c} Q_1 = t_i \\ Q_2 = t_g \\ \vdots \\ Q_{k-1} = t_j \end{array} \right\}$$

3.4 Modelling the interaction

3.4.1 Aim of the model

The predictor should be able to use the statistical model of the interaction, obtained analysing various runs of the same interaction model, to compute the probability distribution of terms for a variable Q_k, given the current state of the interaction.

In the design of the model we should not assume any specific ontologies for the other peer, but rely only on the peer's own: for example, the other peer in an interaction could be a human, without a specific and formal ontology. However, the terms in the received messages are first mapped into terms of the peer ontology: these mapped terms are the ones used to create the model.

3.4.2 Assumptions

The founding assumption, as seen before, is that the same interaction model is repeated when similar situations or tasks occur: in OpenKnowledge, for example, a vendor peer can subscribe to a purchase interaction model and be asked to take part in the interaction every time a potential buyer subscribes to the same interaction model.

Following this assumption, we make four more assumptions that provide the basis for creating the model:

- terms in received messages have a prior probability distribution,

- terms in received messages may have a posterior probability given previous messages and constraints,
- terms in received messages have ontological relations with terms in the agents ontology,
- terms in received messages may have ontological relations with terms in other messages and constraints.

We now Analise more in detail these assumption to verify whether they are reasonable.

Terms in received messages have a prior probability distribution

Within a specific type of interaction, some terms appear more frequently than others. The frequency of the terms depends on two factors:

1. the interaction itself. Different interaction models are used for different purposes. For example, peers using an interaction for purchases will likely use terms related to this task. Interactions can be more specific than others, and this is reflected in the distribution of terms, being narrower in the more specific ones.
2. how the various peers taking part in the interactions instantiate the variables. The frequency of terms reflects "community" needs or desire. These frequencies may change over time, as new needs or ideas appear. Using the Go ogle Trend tool[1], it is possible to verify how many queries for particular terms are made by people in different parts of the world. For instance, queries about Apple phone started nearly suddenly at the beginning of 2007, as Figure 3.5 shows. Figure 3.6 shows how the amount of queries about B&B fluctuates periodically: there is a peak (narrower in Italy than in the world) of requests in summer, and a decrease in winter. Moreover, while the amount of world queries remains similar in the same seasons of different years, the Italian graph shows that the number of requests increases every year.

This hypothesis does not require any further assumptions about relations between the terms in the interaction: it relies only on the wider context of the interaction and of the community in which it is used. It assumes that the other peers, when taken as a community, satisfy constraints according to some distribution, and that requests are not all equally likely. It also does not assume any structured ontology on the side of the peer that creates the model.

[1] `http://www.google.com/trends`

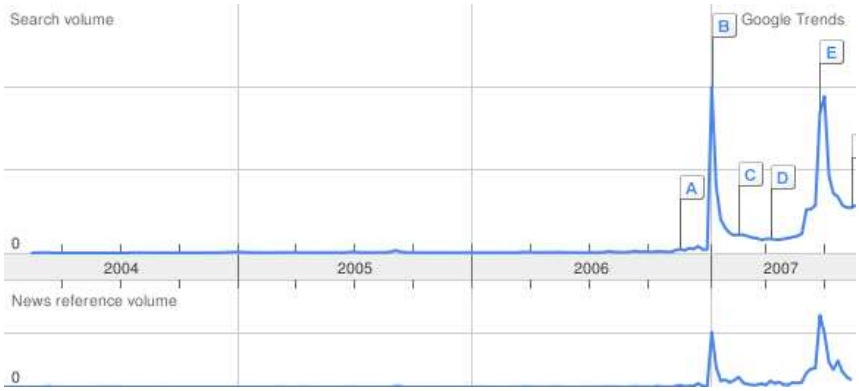

Figure 3.5: *Distribution of* GOOGLE *queries about iPhone*

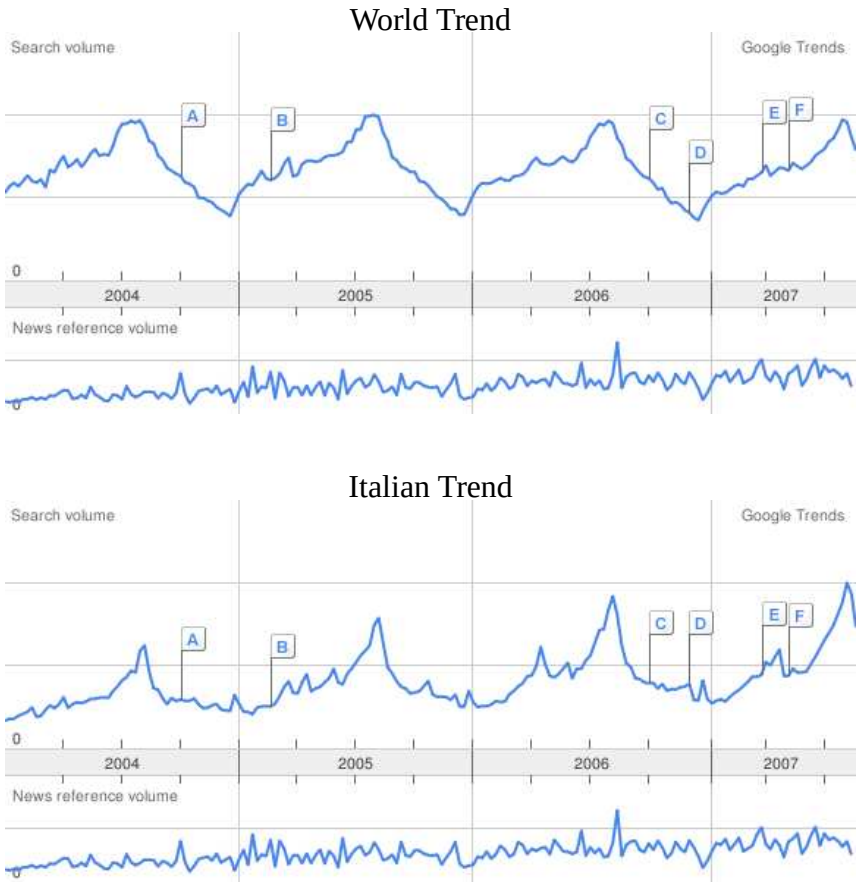

Figure 3.6: *Distribution of* GOOGLE *queries about B&B*

Terms in messages may have posterior probabilities given previous sent or received messages and constraints

This assumption relies on the belief that the current state of an interaction depends on the value of some previous states. The number of previous states taken into account is usually a parameter of the system: the influence of previous states decreases with temporal distance.

It does assume a relation between terms in a dialogue, but the relations are not made explicit: it is only possible to verify that given one term in a specific point of the interaction, another term is more or less frequent. If the interaction model in Figure 2.5 is used for renting a car, then terms like *hotel* or *B&B* will not appear in the offers from the supplier, while terms like *van* or *compact car* will appear more likely. However, it is not possible to know if there is some ontological relation between the terms: it is just assumed that high conditional frequency implies a relation.

Terms in messages have ontological relations with terms in the agent's ontology

This assumption relies on the idea that terms in messages will often belong to the same class. For example, the supplier may verify that the terms received in all the requests are always subclasses of its own classes *"lodging"* or *"flight"*. This information is an abstraction of the term frequency discussed above: it says that the term belongs to a set with a certain probability. The set is the one obtain satisfying the relation with the ontology: if the relation is $subclass(Product, "lodging")$ then *Product* can be any of the subclasses of *"lodging"*. It does not specify which subclass: any of them can be the right one, but it include also terms that have not appeared yet in the performed interaction, increasing its flexibility.

Terms in messages may have ontological relations with terms in other sent or received messages and constraints

The ontological relations can also be verified between terms in a variable and the content of variables both in previous messages and constraints, making the relations between terms explicit. In the example scenario, the customer may verify that the terms appearing in the proposals sent by the supplier are frequently subclasses of the term in its own request: the proposal *hostel* is a subclass of the request *accommodation*. This information is an abstraction of the conditional frequency discussed above, as it makes explicit the relation that is expressed in the conditional formula: the relation assigns the frequency to all the terms that satisfy the relation, given the value of the

other variable.

However, the peers involved in the interaction may have different ontologies, and one of the peers may lack in its ontology the relation that the other peer's ontology has. Moreover, one peer may find relations even when there are none, obtaining a "over fitting" of the relations.

3.4.3 Mapping the assumptions to LCC interaction models

We have repeated that the content of messages comes from heterogeneous *sources*, such as peers in the OpenKnowledge framework or services in BEL workflows. A source is responsible for the introduction of terms related to the interaction and failure to do so disrupts the communication. If the travel agency peer in out example, after being asked for an accommodation, satisfies the constraint `refine(Product, List Refined)` with a choice of possible types of coffee, then the communication loses meaning. Intuitively, sources fall into three main categories:

- *Purely functional*: given a set of parameters, they always return the same values: for example $multiply(X,Y,Z)$ is supposed to always unify the variables with the same numbers.

- *Purely "preference-based"*: they collect requests from users and their possible values can differ every time. In the example, the constraint `want(Product)` is preference-based; each peer will satisfy it according to its tastes and needs. Overall, the variables in preference-based sources will have a (unknown) distribution. These distributions may change with time, depending on general shifts of "tastes" and "needs" (fashions, trends, fads, ...) or the heterogeneity in the peer group composition. A distribution can be more or less skewed: it can be a uniform or it may follow a power-law distribution.

- *Mixed*: they can be mainly functional, but the results may change depending on external factors (availability, new products appearing on the market, etc), or can be mainly preference-based, but constrained by some other parameters. In the example, the constraint `refine(Product,List Refined)` is mainly functional, as it returns the list of possible subclasses of a term if the query can be refined. The list of terms can however change depending on the specific peer and with time.

A purely functional source can be guessed when the function is ontological, that is when it returns terms that are ontologically related to the input term: for instance, they can be its subclasses, or its siblings, or its instances, or its properties. The hypotheses can be verified comparing the guesses with the feedback from the ontology matching process. For the purely preference based, it is possible to count the frequencies of the terms and learn their prior probability distribution. For the mixed, it is possible to use a mix of hypotheses and counting the frequencies. Sometimes the ontology of the peer does not allow him to formulate the correct ontological relation (because the ontology is structured differently from the agent that introduced the term), but it is still possible to count the conditional frequencies, modelling the relation from a purely statistical point of view.

3.5 Goals of prediction

As described in Section 3.3, the predictor provides a probability distribution for the terms that can appear in a particular message during an interaction, given the previously exchanged messages and the history of similar interactions. The probability distribution can be used to select the terms that are more related to the current interaction, excluding those that are not. The selection can be used to improve efficiency, reducing the number of operations required to find the mapping. It can also be exploited to reduce ambiguities in the mappings: when the matchers are unable to distinguish between equally likely correspondences $t_j \ldots t_k$ for a foreign term w_i, the terms that are unrelated to the interaction can be excluded.

This section explains how the objectives of the thesis, that is improving efficiency while maintaining or improving recall and precision of an ontology matcher can be reached by using the results of the predictor. It also shows how it could be possible to use the predictions to provide the basis for extending the agent's ontology.

3.5.1 Predicting for efficiency

The knowledge of the probability distribution of a variable Q_k can be used to select a subset $\Lambda \subseteq O$ of terms likely to appear in it. This set Λ, and not the whole ontology, becomes the set T of terms to compare in Function 2.3 improving both the efficiency and the results of the ontology mapping systems, and making it more feasible to be performed at run-time.

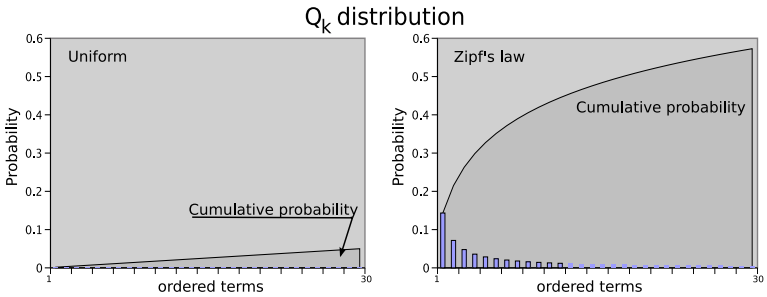

Figure 3.7: *Uniform and Zipf's law distributions*

Assuming the knowledge of the probability distribution defined in Equation 3.4, and assuming that the matching term t_m exists (as we have stated in Section 3.3), the probability that the correct matching term t_m belongs to a set Λ is:

$$P(t_m \in \Lambda) = \sum_{t_i \in \Lambda} P(Q_k = t_i | IM_{state}, M)$$

To select the terms to insert in Λ, it is necessary to set a threshold $\tau \leq 1$ for $P(t_m \in \Lambda)$. If the list Ω contains the terms ordered from the most to the least probable, then this means solving the equation in n:

$$\tau \leq \sum_{j=1}^{n} P\left(t_j\right) \quad t_j \in \Omega \tag{3.5}$$

That simply means taking the first n most likely terms until their cumulative probability is equal or greater than τ. For $\tau = 1$, then $\Lambda \equiv O$, while for $\tau < 1$ the size $|\Lambda|$ depends on the probability distribution. For a uniform distribution it will be directly proportional to τ, while for a skewed distribution, it can be $|\Lambda| \ll \tau |O|$: it becomes useful to trade off between the size of the set Λ and the probability of finding the correct correspondence.

As shown in Figure 3.7, if the probability distribution of the terms is uniform, then $p(t_m \in \Lambda)$ will be proportional to $|\Lambda|$. For example, if $|O| = 1000$, then $P\left(Q_k = t_j\right) = 0.001$ for $\forall t_j \in O$. Setting $|\Lambda| = 800$ yields $P(t_m \in \Lambda) = 0.8$, and there is no strategy for choosing the elements to add to Λ. Instead, if the probability is distributed unevenly, we can keep the most likely terms discarding the others, maintaining at the same time a high probability τ of finding the correspondence t_m in smaller Λ. For example, suppose that $P\left(t_j\right)$ is distributed approximately according to Zipf's law (an empirical law that states that the probability of an item is inversely proportional to its rank):

$p(k; s; N) = \frac{1/k^s}{\sum_{n=1}^{N} 1/n^s}$

where k is the rank of the term, s is a parameter (which we set to 1 to simplify the example), and N is the number of terms in the list of items. The probability of finding t_m becomes:

$p(t_m \in \Lambda) = \frac{\sum_{k=1}^{|\Lambda|} 1/k}{\sum_{n=1}^{|O|} 1/n}$

for $|O| = 1000$, then $P(t_m \in \Lambda) = 0.70$ for $|\Lambda| = 110$ and more remarkably $P(t_m \in \Lambda) = 0.5$ for $|\Lambda| = 25$, as shown in Figure 3.7.

Therefore, given a probability distribution for the terms, it is possible to trade off a decrement in the probability of finding the matching term t_m in Λ with an important reduction of comparisons made by the oracle.

If the oracle cannot find any matching inside the suggested set Λ, it can move to consider a wider set - in the worst case the whole ontology. Given that τ is the threshold for the cumulative probability of terms in Λ, the average number of evaluated hypotheses will be:

$$E[nr\,eval\,hp] = E[|\Lambda|] + (1-\tau)(|O| - E[|\Lambda|])$$

where the operator $E[X]$ is the expected value of a random variable X. In the example seen above, where terms are distributed according to Zipf's law and τ is set to 0.7, then:

$$E[nr\,eval\,hp] = 110 + 0.3 * (1000 - 110) = 377$$

instead of 700.

3.5.2 Predicting for recall

Recall, as defined in Section 2.6, is the ratio between the number of found correspondences and the total number of possible ones, and when real world ontology mapping systems are applied to real world scenarios, precision is fairly high, but recall is often low ($\sim$ 30%) [17]. This usually depends on lack of information about the relation between the term to map and terms in the agent's ontology. The information about the relations, as said in Section 2.6, can be found in the syntactic structure of the term (similar strings), in the ontology structure (similar position in the two ontologies), or implicit in the meaning of the terms. In many case finding this relation requires too much background knowledge or too much domain specific knowledge and the existing bridge between two terms is rejected, lowering the recall rate.

If we do not have enough information to identify the relation between a foreign term and a local term, then this means that all the terms are nearly equiprobable. The proposed system provides, given the current state of the interaction and the history

of previous runs of the same interaction, a probability distribution for the value of Q_k. Given the probability distribution, different from the uniform distribution we have seen before, we are less uncertain about the real value of Q_k: we have therefore more information. This additional information comes simply from having repeated the interaction, and knowing therefore what to expect.

This is an improvement over the situation described by Equation 2.4, that stated that in the classical approach an ontology matcher starts its work considering all the terms equally probable.

3.5.3 Predicting for precision.

As defined in Section 2.6, precision is the ratio between the number of correct correspondences among those found and the total number of found ones. Precision is low when an ontology mapping system maps many foreign terms w_i to wrong terms in the agent's ontology. This is often due to lack of available information that can disambiguate between two (or more) possible correspondences. For example, if only the string similarity is used, then the term *"cars"* has the same normalised edit distance of 0.25 with the terms *"car"*, *"cans"*, and *"cart"*.

The context can provide the information necessary for the disambiguation, suggesting the terms most likely given the state of the interaction: if the interaction is about renting a car, then the most likely term for the matching is *"car"*, and the rest can be discarded.

3.5.4 Predicting for extending ontologies

The three assumptions we made for the system are that the correspondent term t_m exists, the terms in messages have an ontological relation with terms in the peer's ontology and they may have relations with terms in previous messages in an interaction.

These assumptions can also drive the extension of an ontology. If the predicted content for Q_k has a consistent relation $rel(Q_k, Q_{k-i})$ with a previous variable Q_{k-i}, or a relation $rel(Q_k, e_j)$ with a term e_j in the ontology, but the ontology matcher cannot find the corresponding term t_m in the ontology because the term is missing, then this can be an indicator that there is an important term, referred to as w_i in other ontologies that should be added to the ontology and should be in relation $rel(Q_k, Q_{k-i})$ or $rel(Q_k, e_j)$ with the other term.

For example, when Q_{k-i} is *"accommodation"*, sometimes Q_k has the unknown

value of "*residence*": the predictor may suggest that the content of Q_k is a subclass of "*accommodation*", but the ontology matcher fails to find the correspondence. Over time, the repeated failure can be provide an indication for the curator of the customer ontology that she should add a new term corresponding to *"residence"* as a subclass of *"accommodation"*.

3.6 Summary

Ontology mapping systems usually do not consider the context within which the matching is performed. This means that before applying the matchers, all correspondences are equiprobable. However, if we use an ontology mapping system to dynamically map terms in an interaction, we can assume that terms in messages appears with different frequencies. These frequencies are influenced by the specific context of the interaction, by the previously exchanged messages and by the community of participants in the interaction.

By analysing similar interactions it is possible to obtain a model that can be used to compute the distribution of probabilities of terms in the messages of an interaction. These probability distributions can be used to predict the most likely terms in a message, focussing computationally expensive ontology matching activities on them and improving efficiency. They can also be used as additional information provided to the matcher, increasing recall (usually low because of lack of domain specific knowledge) and precision (by removing ambiguities).

Chapter 4

Implementation of the Predictor

4.1 Introduction

While the previous chapter describes the assumptions and the goals of the proposed solution, this chapter presents the architecture and the functioning of the predictor.

In the proposed architecture, the predictor creates the model of an Interaction Model from the mapped terms fed back by the mapping oracle at every run of the interaction. The model is composed of a set of assertions for each variable Q_k in the interaction. An assertion states the frequency with which the terms used for Q_k have appeared in a specified set of terms that share the same property. The set can either be defined by an explicit list or by an ontological relation between the variable and another term. Section 4.3 describes the model and how it is updated.

When the predictor is invoked for a variable Q_k during a run of an interaction, it selects and instantiates the assertions for the variable, and then computes the probability distribution of all the terms in the peer's ontology, passing it to the oracle. Section 4.4 describes, together with an example, how assertions are selected and instantiated and how their frequencies are combined to yield the probability of a term.

4.2 Architecture

As we have stated before, the aim of the system is to exploit the repetitions of similar interactions in order to predict the content of received messages in future interactions. The predictor works in two phases, linked by a feedback loop as shown in Figure 4.1:

model creation: the predictor uses the correspondences found and fed back by the

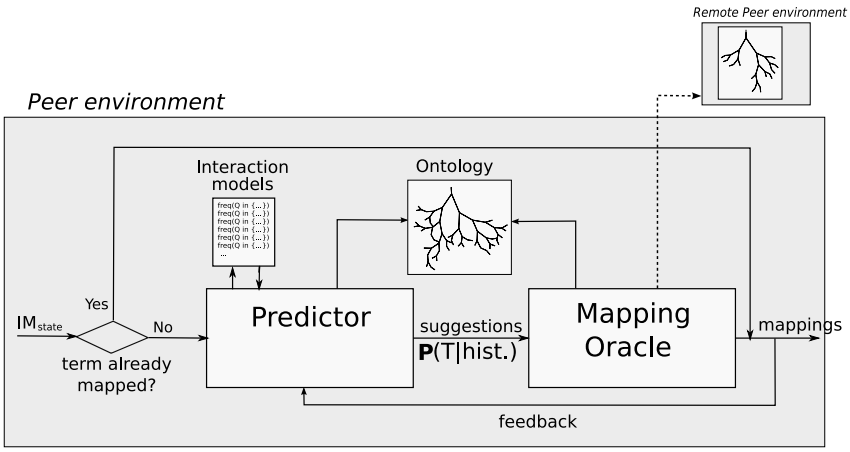

Figure 4.1: *The predictor feeds suggestions to the mapping oracle, that feeds back the correct correspondences (when possible)*

oracle and the peer's ontology to create and update the model,

prediction: it is performed when there is the need to map the content of a variable; the result is a probability distribution for all the terms in the peer's ontology given the past repetition of the interaction model and the current state of the run.

The oracle receives the probability distribution computed by the predictor, and uses it:

- to prioritise the comparison between the foreign term w_i in the message and the terms in the peer ontology
- as additional information, based on the context of the interaction, about the correspondences

The oracle may use the external ontology that defines w_i, depending on the algorithm it uses, but it is irrelevant for the functioning of the system. The best matching found by the oracle is then fed back to improve the model for the particular interaction.

4.3 Model creation and update

The predictor receives the current model of the interaction M, the peer's ontology O, the current state of the interaction IM_{state} and returns an updated version of the model M':

$$update : M \times O \times IM_{state} \rightarrow M'$$

4.3.1 Model representation

This solution, that I first suggested but did not evaluate in [5] and then presented more thoroughly in [6], is a statistical model M of the interaction IM in which the properties of entities appearing in the random variable Q_k in different runs of the same interaction model are counted and stored in a set $\mathbf{A}$ of *assertions*:

$$M = \langle IM, \mathbf{A} \rangle$$

An assertion $A \in \mathbf{A}$ about a random variable Q_k appearing in a clause relative to a role r keeps track of the frequency f with which, given a condition ζ, the entity has been part of a set Ψ defined by some properties in the encountered dialogues:

$$A = \langle id, r, Q_k, \Psi, \zeta, f \rangle \tag{4.1}$$

The condition ζ can be empty (ε) or can specify the value or a property of another variable, Q_j/t_g. The set Ψ can be specified as an explicit list of terms $\{t_1, ..., t_n\}$, or with a *set builder* formula $\{x|\phi(x,e)\}$, where $e \in O$ and O is the peer's local ontology. The explicit list means that the terms in it have appeared, in total, f times in Q_k. The formula means that the relation $\phi(x,e)$ between the term x in Q_k and another entity e has been verified f times: the set Ψ includes all the terms whose property is in relation ϕ with e. The relation is an ontological relation (*subClass*, *superClass*, *siblingOf*, *domainOf*, *rangeOf*); the entity e can be either a term from the agent ontology, or a variable in a previous message in the interaction. The possible types of assertions are listed in Table 4.1.

The available ontological relations depend on the expressivity of the ontology used by the agent: if it is a simple list of terms, then no relations can be found, if it is a taxonomy then it is possible to find subsumption relations, if properties are included then range and domain relations can also be identified. This can be an incentive to develop rich ontologies, as they allow for more detailed relations to be found.

4.3.2 Creating and Updating the Model

Assertions are created and updated every time an interaction model is executed. The predictor works inside the agent's environment, and therefore works only with terms from the local ontology. It receives the translated version of the messages as feedback from the mapping oracle, and then analyses the local terms of the variables in the messages in order to create and compute the assertions, according to different analysis

Frequency of terms:

$\langle j, role, Q_{ki}, \{t_q\}, \varepsilon, f \rangle$

Assertions can be about the frequency of the entities in an argument, disregarding the content of other variables in the dialogue.

For example: $\langle 1, customer, Proposal, \{b\&b\}, \varepsilon, 6 \rangle$.

Conditional frequency of terms:

$\langle j, role, Q_k, \{t_q\}, Q_i = t_h, f \rangle$

More precise assertions can be about the frequency of an entity given the content of previously encountered variables.

For example: $\langle 1, customer, Proposal, \{b\&b\}, Product = "accomodation", 4 \rangle$.

Frequency of relations with terms in other variables:

$\langle j, role, Q_k, \{X \mid rel(X, Q_i)\}, \varepsilon, f \rangle$

They can regard the relation with an argument of another variable E_k in the interaction model.

For example: $\langle 1, customer, Proposal, \{X : subclass(X, Product)\}, \varepsilon, 24 \rangle$

Frequency of relations with terms in ontology:

$\langle j, role, Q_k, \{X \mid rel(X, t_k)\}, \varepsilon, f \rangle$

They can be about an ontological relation between the entity in the argument and an entity t_k in the agent's ontology.

For example: $\langle 1, customer, Proposal, \{X : subclass(X, "product")\}, \varepsilon, 24 \rangle$

Table 4.1: *Types of assertions*

strategies that follow from the assumptions listed in Section 3.4. The strategies search for different properties of the terms:

- Terms that appear in a variable are counted. Their property is simply being identical to a term already encountered or being a newly met term. An assertion for each term is generated, and every time the same term reappears the frequency of the assertion is increased.

- Terms that appear in a variable are counted, but assertions are generated with a condition ζ about the value of a previous variable. In this case the property of the terms is being identical to a previous term (or being new) and following the same term as the previous ones (or a new one). Every time the same term reappears, satisfying the condition ζ by following the same term in a previous variable, the frequency of the assertion is increased. For example, if the translated value of the variable `Proposal` is "*hotel*" and the translated value of `Product` in the previous

message is "*accommodation*", then an assertion about this case is created:
$\langle ..., customer, \texttt{Proposal}_k, \{\text{“}hotel\text{”}\}, \texttt{Product} = \text{“}accommodation\text{”}, 1\rangle$
If the same combination of terms appears in future interactions, the frequency of this assertion will be increased. The maximum distance between the variables is a parameter of the strategy.

- Different ontological relations between the terms in a variable and terms in the peer's ontology are checked. An assertion for each satisfied relation is generated, and its frequency is increased every time the same relation is satisfied. All the terms in the set Ψ of the assertion share the same relation with the term in the peer's ontology.
  More formally, the system searches the terms $x_1, x_2, ... \in O$ for which the following relations hold:
  $O \vdash \phi_1(q_k, x_1), O \vdash \phi_2(q_k, x_1), O \vdash \phi_3(q_k, x_1), ...$
  $O \vdash \phi_1(q_k, x_2), O \vdash \phi_2(q_k, x_2), O \vdash \phi_3(q_k, x_2), ...$
  The relation between the variable Q_k and the found term x_i, $\phi_j(Q_k, x_i)$, is stored if new or updated otherwise. In practice, the most useful relation to verify and store is the one about the term that generalises the value of Q_k : knowing the a variable always contains objects of a certain class is similar to finding by induction the type of the variable and help to predict the possible content of instances of the same variable in future interactions.
  For example, if "*hotel*" is the translated value for the variable `Proposal` in the received `offer()` message (see Figure 2.7), then the system tries to find its superclass in the agent's ontology. The resulting assertion is about the set of terms that are subclasses of the found superclass ("*accommodation*" in this case):
  $\langle ..., customer, \texttt{Proposal}_k, \{X : subClassOf(X, 'accommodation')\}, \varepsilon, 1\rangle$
  In future execution of the same interaction model, if the value of `Proposal` is translated into another subclass of *"accommodation"*, such as *"b&b"*, then the frequency of the assertion is increased

- Different ontological relations between the terms in the variable and the *mapped* value of previous variables are checked. An assertion for each satisfied relation is generated, and its frequency is increased every time the same relation is satisfied. The terms in the set Ψ of the assertion all share the same relation with another variable in the same interaction model.
  More formally, the system tries to prove which of the following relations hold,

given the agent's ontology as set of axioms:
$O \vdash \phi_1(q_k, q_{k-1}), O \vdash \phi_2(q_k, q_{k-1}), O \vdash \phi_3(q_k, q_{k-1}), ...$
$O \vdash \phi_1(q_k, q_{k-2}), O \vdash \phi_2(q_k, q_{k-2}), O \vdash \phi_3(q_k, q_{k-2}), ...$
The holding relations are stored, and if already encountered are increased. For example, if the value of `Proposal` translates into *"hotel"*, and that of `Product` into *"accommodation"*, the system tries to prove different ontological relations between the terms: it checks if *"hotel"* is a superclass, a subclass, a sibling, a property of *"accommodation"*. The correct relation between the variables is stored:
$\langle ..., customer, \texttt{Proposal}_\texttt{k}, \{X : subClassOf(X, \texttt{Product})\}, \varepsilon, 1 \rangle$
When the same relation reappears in another run of the interaction, for example because `Product` is *"car"* and `Proposal` is *"van"*, the frequency of the assertion is increased. The distance up to which search for relations is a parameter of the strategy.

Table 4.2 shows the possible model for the content of the variable $Proposal_k$ in the interaction model in Figure 2.5, that the customer peer may have created after having executed the interaction a number of times with different types of service providers.

4.3.3 Example of creation and update

In our example the customer peer uses the same interaction model to perform different tasks, such as booking car rentals and accommodations, dealing with various suppliers. The first interaction is depicted in Figure 3.1: the customer asks "*accommodation*", and the supplier, possibly a travel agency or an hotel agent, replies with "*hostel*", that is rejected, and then with "*bed&breakfast*". As we have seen in the figure, the term in the second proposal must be mapped to "*b&b*" in the customer ontology. The predictor module receives as feedback the satisfied constraints and sent messages with the translated terms. In this case, the predictor receives:

```
1) constraint:  want("accommodation"),
2) messagein:  offer("hostel"),
3) constraint:  acceptable("hostel", "accommodation"),
4) messagein:  offer("b&b"),
5) constraint:  acceptable("b&b", "accommodation")
```

The predictor stores the translated unfolding of the interaction for the length of the run, in order to find relations between the terms in previously received messages. The

constraints are only stored, while received messages are processed and the statistical model is updated.

When the first `messagein` arrives, `offer("hostel")` in this case, the predictor:

1. checks if there is an assertions about the frequency of term *"hostel"*. This is the first time the interaction is used, so there are no assertions, and it creates a new one:
   $A_1 = \langle 1, \textit{customer}, \texttt{Proposal}_k, \{\text{“}\textit{hostel}\text{”}\}), \varepsilon, 1\rangle$

2. checks if there is an assertion about the conditional frequency of "*hostel*" given the value "*accommodation*" in `Product` variable. Because there are no assertion yet, it creates a new one:
   $A_2 = \langle 2, \textit{customer}, \texttt{Proposal}_k, \{\text{“}\textit{hostel}\text{”}\}, \texttt{Product}_1/\text{”}\textit{accomodation}\text{”}, 1\rangle$

3. searches the superclass of "*hostel*" in the peer's ontology, trying to satisfy the relation `superclass(X,"hostel")`, finding "*accommodation*". It checks if there is an assertions about the relation, and as this is the first run it creates a new one:
   $A_3 = \langle 3, \textit{customer}, \texttt{Proposal}_k, \{\texttt{X : subClassOf}(\texttt{X}, \text{“}\textit{accommodation}\text{”})\}, \varepsilon, 1\rangle$

4. tries to prove different relations between "*hostel*" and the terms in variables appearing in previous messages and constraints. In this case, it tries to satisfy:
   `subclass(Proposal, Product), superclass(Proposal, Product), siblingOf(Proposal, Product), propertyOf(Proposal, Product), propertyOf(Product, Proposal)`
   `Proposal` is replaced by "*hostel*" and `Product` is replaced by "*accommodation*", and the only relation that can be proved is `subclass("hostel", Product)`. Being the first interaction, there are no assertions and a new one is created:
   $A_4 = \langle 4, \textit{customer}, \texttt{Proposal}_k, \{\texttt{X :subClassOf}(\texttt{X}, \texttt{Product}_1)\}, \varepsilon, 1\rangle$

When the second `messagein` is fed to the predictor, a similar process takes place. The last two cases are verified again, as "*b&b*" is a subclass of the term "*accommodation*", and therefore the assertions 3 and 4 are updated, increasing their frequency.

If the same interaction is then used in the interaction shown in Figure 2.8 for renting a car, the predictor receives as feedback during the run the following translated interaction events:

```
1) constraint:  want(car)
2) messagein:  offer(compact_car)
3) constraint:  acceptable(car,compact)
4) messagein:  offer(economy_car)
```

`5) constraint: acceptable(economy_car,car)`

When the first messagein event arrives, the predictor:

1. checks if there is an assertions about the frequency of term *"compact_car"*. There is no assertion, so it creates a new one:
   $A_5 = \langle 5, \textit{customer}, \texttt{Proposal}_\texttt{k}, \{\text{“}\textit{compact\_car}\text{”}\}), \varepsilon, 1\rangle$

2. checks if there is an assertion about the conditional frequency of "*compact_car*" given the value "car" in `Product` variable. There is no assertion yet, so it creates a new one:
   $A_6 = \langle 6, \textit{customer}, \texttt{Proposal}_\texttt{k}, \{\text{“}\textit{compact\_car}\text{”}\}, \texttt{Product}_1/\text{“}\textit{car}\text{”}, 1\rangle$

3. searches the superclass of "*compact_car*" in the peer's ontology, trying to satisfy the relation `superclass(X,"compact_car")`, finding "*car*". It checks if there is an assertions about the relation, and as there are none it creates a new one:
   $A_7 = \langle 7, \textit{customer}, \texttt{Proposal}_\texttt{k}, \{\texttt{X : subClassOf}(\texttt{X}, \text{“}\textit{car}\text{”})\}, \varepsilon, 1\rangle$

4. tries to prove different relations between "*compact_car*" and the terms in variables appearing in previous messages and constraints. In this case, it tries to satisfy:
   `subclass(Proposal, Product), superclass(Proposal, Product), siblingOf(Proposal, Product), propertyOf(Proposal, Product), propertyOf(Product, Proposal)`
   `Proposal` is replaced by "*compact_car*" and `Product` is replaced by "*car*", and the only relation that can be proved is `subclass(Proposal, Product)`. An assertion about this relation was created the previous round, and therefore it is only updated:
   $A_4 = \langle 4, \textit{customer}, \texttt{Proposal}_\texttt{k}, \{\texttt{X : subClassOf}(\texttt{X}, \texttt{Product}_1)\}, \varepsilon, 3\rangle$

After 12 runs of the interaction, the resulting model is shown in Table 4.2.

4.4 Prediction of Q_k

The predictor receives the model M for the current interaction, the peer's ontology O and the current state of the interaction IM_{state} and returns the probability distribution for Q_k:

$$predict : M \times O \times IM_{state} \rightarrow \mathbf{P}(Q_k | M, IM_{state})$$

Term frequency

$A_1 = \langle 1, \textit{customer}, \texttt{Proposal}_\texttt{k}, \{\text{“}\textit{hostel}\text{”}\}), \varepsilon, 6\rangle$

$A_8 = \langle 8, \textit{customer}, \texttt{Proposal}_\texttt{k}, \{\text{“}\textit{b\&b}\text{”}\}), \varepsilon, 4\rangle$

$A_5 = \langle 5, \textit{customer}, \texttt{Proposal}_\texttt{k}, \{\text{“}\textit{compact\_car}\text{”}\}), \varepsilon, 3\rangle$

$A_{11} = \langle 11, \textit{customer}, \texttt{Proposal}_\texttt{k}, \{\text{“}\textit{hotel}\text{”}\}, \varepsilon, 6\rangle$

$A_{12} = \langle 12, \textit{customer}, \texttt{Proposal}_\texttt{k}, \{\text{“}\textit{economy\_car}\text{”}\}), \varepsilon, 5\rangle$

Conditional frequencies

$A_2 = \langle 2, \textit{customer}, \texttt{Proposal}_\texttt{k}, \{\text{“}\textit{hostel}\text{”}\}, \texttt{Product}_1/\text{“}\textit{accommodation}\text{”}, 6\rangle$

$A_6 = \langle 6, \textit{customer}, \texttt{Proposal}_\texttt{k}, \{\text{“}\textit{compact\_car}\text{”}\}, \texttt{Product}_1/\text{“}\textit{car}\text{”}, 3\rangle$

$A_9 = \langle 9, \textit{customer}, \texttt{Proposal}_\texttt{k}, \{\text{“}\textit{hotel}\text{”}\}, \texttt{Product}_1/\text{“}\textit{accommodation}\text{”}, 6\rangle$

$A_{10} = \langle 10, \textit{customer}, \texttt{Proposal}_\texttt{k}, \{\text{“}\textit{b\&b}\text{”}\}, \texttt{Product}_1/\text{“}\textit{accommodation}\text{”}, 4\rangle$

$A_{13} = \langle 13, \textit{customer}, \texttt{Proposal}_\texttt{k}, \{\text{“}\textit{economy\_car}\text{”}\}, \texttt{Product}_1/\text{“}\textit{car}\text{”}, 5\rangle$

Ontology-variable frequencies

$A_3 = \langle 3, \textit{customer}, \texttt{Proposal}_\texttt{k}, \{\texttt{X : subClassOf}(\texttt{X}, \text{“}\textit{accommodation}\text{”})\}, \varepsilon, 16\rangle$

$A_7 = \langle 7, \textit{customer}, \texttt{Proposal}_\texttt{k}, \{\texttt{X : subClassOf}(\texttt{X}, \text{“}\textit{car}\text{”})\}, \varepsilon, 8\rangle$

Inter-variables relation frequencies

$A_4 = \langle 4, \textit{customer}, \texttt{Proposal}_\texttt{k}, \{\texttt{X : subClassOf}(\texttt{X}, \texttt{Product}_1)\}, \varepsilon, 24\rangle$

$A_{14} = \langle 14, \textit{customer}, \texttt{Proposal}_\texttt{k}, \{\texttt{X : siblingOf}(\texttt{X}, \texttt{Proposal}_{\texttt{k}-1})\}, \varepsilon, 12\rangle$

Table 4.2: *Statistical model of the context for the customer peer*

The predictor first selects the assertions relative the variable Q_k, then it instantiates the abstract assertions, and finally it combines the frequencies from overlapping assertions.

4.4.1 Instantiating the assertions

The assertions computed using the feedback from the mapping oracle reflect patterns found in different runs of the same interaction model: when the content of a variable Q_k in a new run must be predicted, the set $\mathbf{A}_{Q_k}$ of assertions relative to the variable must be instantiated with the current state of the interaction. The state is given by the unifications of the variables in the messages and constraints encountered up to Q_k: $IM_{state} = \{Q_1/t_i \ldots Q_{k-1}/t_j\}$. The result is the set of instantiated assertions $\mathbf{A}^I_{Q_k}$.

1. some of the conditional assertions in $\mathbf{A}_{Q_k}$ may have a condition ζ not consistent with the current state of the interaction IM_{state}: for example $\langle ..., Q_k, \{t_j\}, Q_{k-1} = t_h, ... \rangle$, when $Q_{k-1} \neq t_h$. These inconsistent assertions are filtered from $\mathbf{A}_{Q_k}$:
   $filter\_inconsistent : \mathbf{A}_{Q_k} \times IM_{state} \rightarrow \mathbf{A}_{Q_k}$
   The filter is done applying to each assertion in $\mathbf{A}_{Q_k}$ the function *verify_inconsistent*:
   $verify\_inconsistent : A \times IM_{state} \rightarrow boolean$
   The function can be expressed in functional, Haskell-like, form:
   $verify\_inconsistent\,(\langle \_, \_, \_, \_, \zeta, \_ \rangle, IM_{state})$
   $\quad \zeta \in IM_{state} \quad = true$
   $\quad otherwise \quad = false$
2. some of the relations in the remaining assertions are about uninstantiated variables. The variables in the relations must be unified with their translated values. This is done by applying to each assertion the function *unify*:
   $unify : A \times IM_{state} \rightarrow A$
   That we can expressed in functional form:
   $unify\,(\langle id, r, Q_k, \{X : rel\,(X, Q_j)\}, \zeta, f \rangle, IM_{state})$
   $\quad Q_j/t_h \in IM_{state} \quad = \langle id, r, Q_k, \{X : rel\,(X, t_h)\}, \zeta, f \rangle$
   $\quad otherwise \quad = Error$

3. some of the assertions, at this point, will be have explicit lists (most of them composed by a single term), while others will define sets through ontological relations between the variable and another term in the ontology. The implicit set must be made explicit computing the relations. This is done applying the function $instantiate-all$ to $\mathbf{A}_{Q_k}$:
   $instantiate-all : \mathbf{A}_{Q_k} \times O \rightarrow \mathbf{A}^I_{Q_k}$
   that applies *instantiate* to each assertion:
   $instantiate : A \times IM_{state} \rightarrow A$
   This function can be expressed in functional form:
   $instatiate\left(\langle id, r, Q_k, \{X : rel\,(X, t_h)\}, \zeta, f\rangle, O\right)$
   $= \left\langle id, r, Q_k, \left\{..., t_g, ...\right\}, \zeta, f\right\rangle \forall t_g : O \vdash rel\,(t_g, t_h)$
   Terms that have already been mapped in previous messages of the same interaction can be removed from the resulting lists: if the foreign term is known, the prediction and mapping phases are bypassed and the term is fed back directly to the modeller.

As anticipated in Section 3.4, assertions about ontological relations create two problems. First, some of the relations can be spurious. Second, some relations may refer to large sets, bringing little information. To deal with the first issue, only relations found in a significant proportion of the cases are taken into consideration. To deal with the second issue, sets larger than a significant portion of the ontology are discarded.

4.4.2 Combining the assertions

The result of the previous steps is a set $\mathbf{A}^I_{Q_k}$ of possibly overlapping sets, each with an assigned frequency. For example:

$$\mathbf{A}^I_{Q_K} = \left\{ \begin{array}{l} \langle ..., Q_k, \{t_1\}, ..., f_1\rangle \\ \langle ..., Q_k, \{t_2\}, ..., f_2\rangle \\ \ldots \\ \langle ...Q_k, \{t_1, t_3, t_5\}, ..., f_n\rangle \end{array} \right\}$$

To obtain a probability of each term $t_1, ..., t_n$ in the agent's ontology the predictor needs to combine the sets and their frequencies. The first issue is how to assign weight to single terms in sets. An initial consideration is that an assertion about ontological relation makes no assumption about the distribution of frequencies of the terms that satisfy the relation: therefore, according to the *principle of indifference*, their frequency can be considered as evenly distributed. From the assertion:

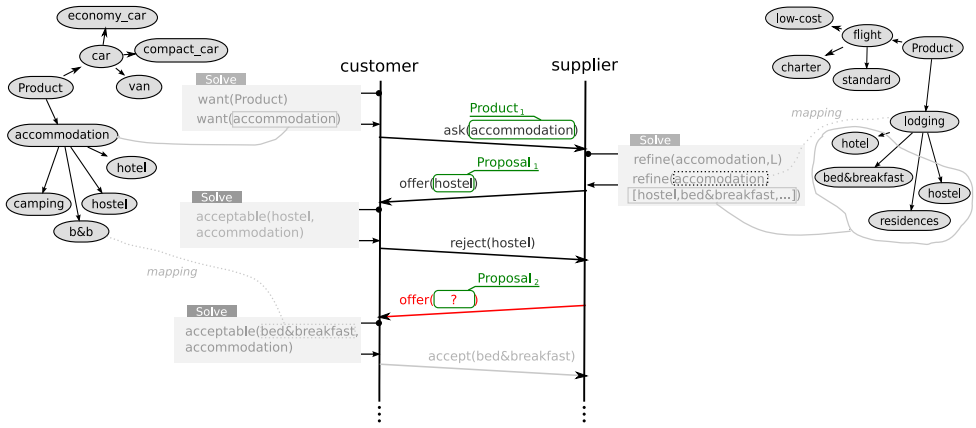

Figure 4.2: *Predicting a variable*

$$A_h = \langle ..., Q_k \{t_1, ..., t_n\}, ..., f \rangle$$

it is possible to obtain a list of assertions about the single terms:

$$A_{h1} = \left\langle ..., Q_k, \{t_1\}, ..., \frac{f}{|\{t_1,...,t_n\}|} \right\rangle$$
$$\vdots$$
$$A_{hn} = \left\langle ..., Q_k, \{t_n\}, ..., \frac{f}{|\{t_1,...,t_n\}|} \right\rangle$$

The result is that the same term t_i may appear in different instantiated assertions, obtained through different strategies (simple frequency, conditional frequency, ontological relations, etc). These frequencies can be summed together and normalised by the frequencies of all the selected assertions $\mathbf{A}^I_{Q_k}$ to obtain the probability of the term t_i:

$$p(Q_k = t_i) = \frac{\sum A_j (t_i \in \Psi)}{\sum_{A_k \in \mathbf{A}} A_k} \tag{4.2}$$

The three Kolmogorov axioms are satisfied:

- $p(Q_k = t_i) \geq 0 \quad \forall t_i \in O$: if a term does not appear in any assertion its probability will be 0
- $\sum_{t_i \in L_a} p(Q_k = t_i) = 1$: the denominator is given by the sum of all the assertions that can appear in the numerator
- the probability of disjoint terms is given by the their sum:
  $p(Q_k = t_i) \cup p(Q_k = t_j) = p(Q_k = t_i \vee t_j) = p(Q_k = t_i) + p(Q_k = t_j)$

4.4.3 Example of prediction

The state of the interaction for the customer peer when it needs to predict the content of $\texttt{Proposal}_2$ in the interaction shown in Figure 4.2 is:

$$IM_{state} = \left\{ \begin{array}{l} \texttt{Product}_1 = \text{"}accommodation\text{"} \\ \texttt{Proposal}_1 = \text{"}hostel\text{"} \end{array} \right\} \tag{4.3}$$

Given that the model M of the interaction model is shown in Table 4.2, and that $k = 2$ (we have recursed once), in order to compute the probability distribution

$$\mathbf{P}\left(\texttt{Proposal}_2 | IM_{history}, IM_{state}\right)$$

the customer peer must:

1. drop the conditional assertions whose condition ζ does not correspond to the current state of the interaction; so assertions A_6 and A_{13} are dropped because their condition $\texttt{Product}_1 = \text{"}car\text{"}$ is inconsistent with the state in Equation 4.3,

2. unify the variables in relations with the current state of the interaction; $\texttt{Product}_1$ in A_4 is replaced with *"accommodation"* and $\texttt{Proposal}_{\texttt{k}-1}$ in A_{14} is replaced with *"hotel"*, obtaining:
   $A_4 = \langle 4, customer, \texttt{Proposal}_2, \{X\texttt{:subClassOf}(X, \text{"}accomodation\text{"})\}, \varepsilon, 24\rangle$
   $A_{14} = \langle 14, customer, \texttt{Proposal}_2, \{X\texttt{: siblingOf}(X, \text{"}hotel\text{"})\}, \varepsilon, 12\rangle$

3. compute the relations in the assertions using the peer's ontology in Figure 4.2, obtaining sets of terms; assertions A_4, A_{14}, A_3, A_7 become:
   $A_4)\langle 4, customer, (\texttt{Proposal}_2 \in \{\text{"}hostel\text{"}, \text{"}hotel\text{"}, \text{"}b\&b\text{"}, \text{"}camping\text{"}\}, \varepsilon, 24\rangle$
   $A_{14})\langle 14, customer, \texttt{Proposal}_\texttt{k}, \{\text{"}hotel\text{"}, \text{"}b\&b\text{"}, \text{"}camping\text{"}\}, \varepsilon, 12\rangle$
   $A_3)\langle 3, customer, \texttt{Proposal}_\texttt{k}, \{\text{"}hostel\text{"}, \text{"}hotel\text{"}, \text{"}b\&b\text{"}, \text{"}camping\text{"}\}, \varepsilon, 16\rangle$
   $A_7)\langle 13, customer, \texttt{Proposal}_\texttt{k}, \{\text{"}economy\_car\text{"}, \text{"}compact\_car\text{"}, \text{"}van\text{"}\}, \varepsilon, 8\rangle$

4. drop the assertions whose set Ψ is larger than a certain proportion of the ontology, as they do not carry useful information. In this case, none is dropped.

In the example, the denominator of the formula is obtained summing the frequencies of the remaining assertions $\mathbf{A^I} = \{A_{1-5}, A_{7-12}, A_{14}\}$. In order to compute the probability that the concept in $\texttt{Proposal}_2$ is the term *"hotel"* , we select the assertions whose set contains the term *"hotel"*, obtaining assertions A_3, A_4, A_9, A_{11}, A_{14}. The assertions A_3, A_4, A_{14} contain more than one element, and therefore the frequency assigned to *"hotel"* is computed dividing the frequency assigned to the set by the size of the set to obtain the following:

$$P(Proposal_2='hotel')=\frac{4+6+6+6+4}{92}=\frac{26}{92}=0.282$$
$$P(Proposal_2='hostel')=\frac{6+6+6+4}{92}=\frac{22}{92}=0.229$$
$$P(Proposal_2='b\wedge b')=\frac{4+4+6+3+4}{92}=\frac{21}{92}=0.228$$
$$P(Proposal_2='camping')=\frac{6+3+4}{92}=\frac{13}{92}=0.141$$
$$P(Proposal_2='economy_{car}')=\frac{5}{92}=0.054$$
$$P(Proposal_2='hotel')=\frac{3}{92}=0.032$$

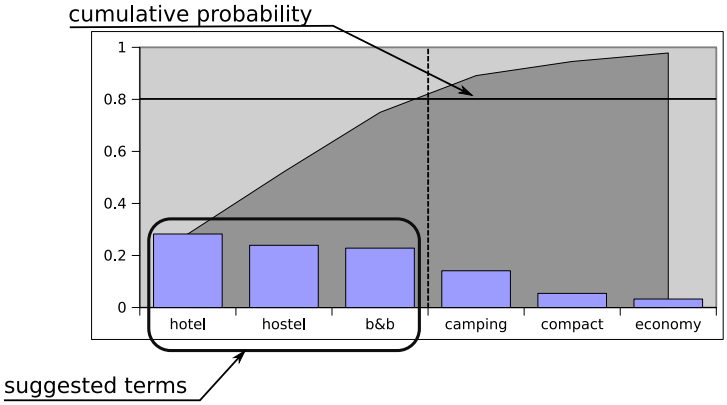

Figure 4.3: *Probability distribution for variable* $\texttt{Proposal}_2$

$P(\texttt{Proposal}_2 = \text{"}hotel\text{"}) = \frac{16/4+24/4+6+6+12/3}{6+6+4+3+5+6+6+4+24+12+16} = \frac{26}{92} = 0.282$

The complete distribution of variable $P(Proposal_2|IM_{history}, IM_{state})$ is shown in Figure 4.3.

4.5 Summary

In this chapter we presented the architecture and the functioning of the predictor. The predictor creates a model for a variable Q_k in an interaction model from the feedback obtained by the mapping oracle. The model is composed of assertions about the frequency with which the term corresponding to the entity in Q_k appeared in a particular set, defined either by an explicit list or by a set builder formula. An assertion can be about the frequency with which a term has appeared in Q_k (possibly given other terms in previous messages), or about the frequency which an ontological relation between the content of the variable Q_k and either a term in the ontology or another variable Q_{k-j} has been found. The model is used to compute the probability distribution of terms for the variable Q_k selecting the assertions that are consistent with the current interaction run, instantiating those defined by formulas and combining them for each

term.

The statistical framework presented in this chapter might resemble a Hidden Markov Model, and in fact it was partially inspired by the intuitive idea behind it. However, the use of ontological relations between variables, that, as we will see in the next chapter, represents one of the strength of this work, cannot be represented using a Markovian model. Moreover, it violates the Markovian assumption that requires that the current state depends only on a finite history of previous states. Choreographies can be of any length, and, because some sections of the choreographies may be repeated, their runs can be of different duration each time. Therefore a variable in a message can have an ontological relation with another variable in a previous message at an arbitrary distance.

Chapter 5

Evaluation

5.1 Introduction

In Chapter 3 we have introduced and explained the idea of using the history of previous interactions and the state of the current interaction in order to compute the probability distribution of the terms in a particular message in a defined interaction between agents. In Chapter 4 we have provided an implementation for the predictor, based on collecting statistics on the content of messages.

We now have to verify its functionality and its usefulness, answering two main questions: 1. Does it work? 2. Is it useful? The first question, answered in Sections 5.3.2 and 5.3.3, requires verifying whether the predictions, i.e. the computed probability distributions, are correct. The probability distribution computed by the predictor is correct when it reflects the real probability distribution of the messages' content. Another element to verify is the robustness of the predictor when the community of users changes, influencing the real probability distribution of the content.

The second question, mainly answered in Section 5.4, requires ascertaining whether the use of the predictor improves the performance of an ontology matcher, measured in computational time complexity, precision and recall.

5.2 General Testing Methodology

One way of testing my system is through real interaction scenarios, using real ontologies and real workflows for the dialogues, but since these are scarce this would cover only part of the testing space, without having the possibility to vary parameters in order to verify the effects.

$$
\begin{array}{l}
\mathtt{a(r8a(O),I)}:: \\
\quad \mathtt{m_1(X,P)} \Rightarrow \mathtt{a(r8b,O)} \leftarrow \kappa_1\mathtt{(P,X)} \\
\quad \mathtt{then} \left(\begin{array}{l} \mathtt{m_2(Y)} \Leftarrow \mathtt{a(r8b,O)} \\ \mathtt{or} \\ \mathtt{m_3(M)} \Leftarrow \mathtt{a(r8b,O)} \end{array} \right) \\
\\
\mathtt{a(r8b,O)}:: \\
\quad \mathtt{m_1(X,P)} \Leftarrow \mathtt{a(r8a,I)} \\
\quad \mathtt{then} \left(\begin{array}{l} \mathtt{m_2(Y)} \Rightarrow \mathtt{a(r8a,O)} \leftarrow \kappa_2\mathtt{(P,X,Y)} \\ \mathtt{or} \\ \mathtt{m_3(M)} \Rightarrow \mathtt{a(r8a,O)} \leftarrow \kappa_3\mathtt{(P,X,M)} \end{array} \right)
\end{array}
$$

Figure 5.1: *Interaction model template*

What is important, however, is to verify the ability of the predictor to statistically model the way in which constraints are satisfied given the state of the interaction. And, as we have seen in Section 3.4, the constraints can be *functional*, *preference-based*, or *mixed*. It is thus possible to simulate different real world scenarios using template interaction models executed by dummy peers that can only satisfy constraints according to parametrisable rules and ontologies.

In order to test and evaluate the feasibility and the reliability of the model, we developed a framework that can run different dialogues, analysing the message content in order to create models for the interactions, and then applying them to predict the content of messages in similar interactions.

Interaction Framework

The template interaction models must cover the basic patterns present in interactions. For example, the interaction model in Figure 5.1 can model many different interactions: m_1 can be a request for information X about P (for example, the price of a X), with m_2 being the reply and m_3 being the apology for not knowing the answer. Alternatively, m_1 can be an offer (the product X at price P), with m_2 being the acceptance and m_3 the rejection. By viewing interaction models abstractly we can set up large scale experiments in which we vary the forms of constraints in a controlled way.

The functional constraints are ontological rules, the preference-based constraints return terms according to probability distributions that reflect a distribution of “needs” and “tastes” over a community of peers, and mixed constraints are rules with an ele-

ment of probability.

Ontologies

The ontologies are generated as graphs, composed by a main tree, that corresponds to the class taxonomy plus the instances, and links between the classes that represent the properties. Because it is possible to specify the features of generated ontologies, it is possible to cover a wider space of variations than it by using existing ontologies.

Constraints

Peers introduce terms in interaction models satisfying constraints. As we have seen in Section 3.4, constraints can be:

- *purely functional*: when given the input arguments, the output is always the same. For example, the constraint *multiply(X,Y,Z)* should unify Z always with the same value given the same X and Y
- *purely preference based*: when the output depends only on a probability distribution. For example, the constraint *want(P)* in the example scenario unifies values that reflect the preference of the community of peers that use the interaction
- *mixed*: when the output depends on the input parameters, but it is not deterministic, and the possible set of terms in the output follow a probability distribution

The way constraints are solved is simulated in the agents. In particular, preference based constraints are solved returning terms according to a probability distribution whose parameters can be modified to verify the behaviour of the predictor in different situations. A preference function takes an ordered list of terms $R \subseteq O$, where O is the full ontology, generates a number $0 \leq i \leq |R|$ according to a probability distribution (in the experiments, we used the half-normal distribution) and returns the term at position i inside R.

The width of a Gaussian distribution is given by its standard deviation σ: a higher σ means a more spreaded distribution. Figure 5.2 shows the different probabilities of terms ranking from 0 to 120 when Gaussian distributions with different standard deviations used: with $\sigma = 5$, the term ranked first is twice more probable than the term ranked 40^{th}, while with $\sigma = 25$ the probability remains nearly constant over all the terms. Figure 5.3 shows the distributions obtained calling the preference function over

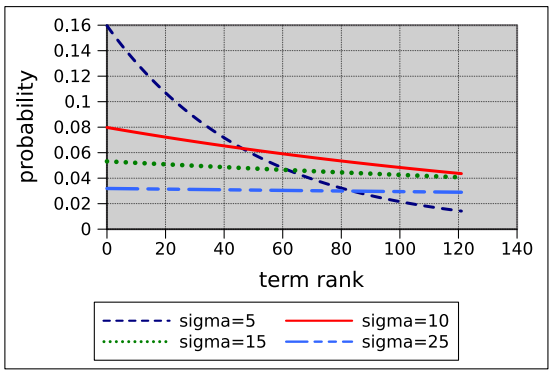

Figure 5.2: *Gaussian distributions with different standard deviations*

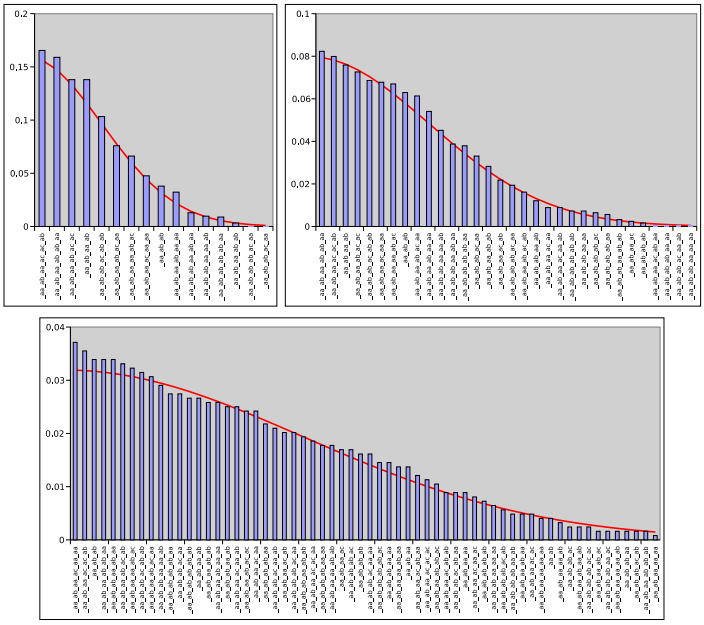

Figure 5.3: *Different preference distributions of terms from a generated ontology*

a thousand times with the same set of terms and first with a standard deviation $\sigma = 5$, then $\sigma = 10$ and finally $\sigma = 25$.

Running the experiments

The experiments consist of running repeatedly (between 200 and 400 times) a number of different interaction models, the constraints of which are satisfied using probability distributions to simulate a large population of agents. Every 10 interactions, a set of performance measures is logged. The performance measures are averaged over a sliding window of 30 interactions.

Each batch of experiments is described in an XML file: the involved agents are

```
<batch>
 <description>use of interaction model 1</description>
 <involved_agent id="tagent1"/>
 <involved_agent id="tagent2"/>
 <experiment id="1">
   <description>Learn the distribution of a variable (with sigma=40)</description>
   <agent_param agent="tagent1" section="general" param="feedback_results" value="true"/>
   <agent_param agent="tagent1" section="randprefs" param="totell" value="{'file':'t1pa', 'sigma':40}"/>
   <institution name="prot1" repeat="200" dumpevery="10">
    <start role="r8a" agent="tagent1">
     <param>tagent2</param>
    </start>
   </institution>
 </experiment>
 <experiment id="2" derived_from="1">
  <description>Learn the distribution of a variable (with sigma=5)</description>
  <agent_param agent="tagent1" section="randprefs" param="totell" value="{'sigma':5}"/>
 </experiment>
</batch>
```

Figure 5.4: *XML file describing an experiment*

listed first, then, for each experiment, the values for parameters are defined (to allow different behaviours in different experiments), and finally it is specified what interaction model must be run with which parameter settings and how many times.

The file shown in Figure 5.4 describes two experiments using the example interaction model in Figure 5.1. The only difference between the two experiment, both involving 200 repetitions of the interaction, is in the variance of the Gaussian distribution: the curve in the first experiment is narrower than in the second.

5.3 Verifying functionality

In this section we evaluate how close the predicted distribution is to the actual distribution of terms. In this experiments I am not concerned with ontology mapping, and therefore the peers share the same ontology. Their goal is only to predict the content of variables in messages before checking them: if the computed distributions are correct, then the peers will often guess the exact term. The suggested set Λ of most likely terms for a variable, described in Section 3.5, is the core criterion used in evaluating the functionality. The average size of the set, the likelihood that the correct term is in the set, and the average rank of the correct term in the set are used as indicator of the ability of the predictor.

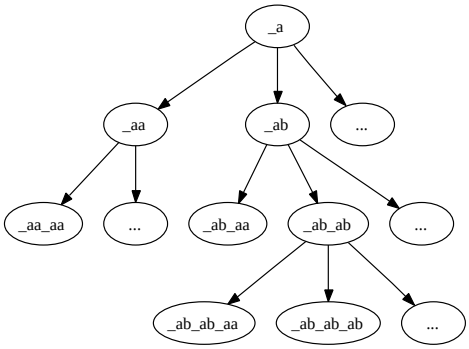

Figure 5.5: *A generated ontology*

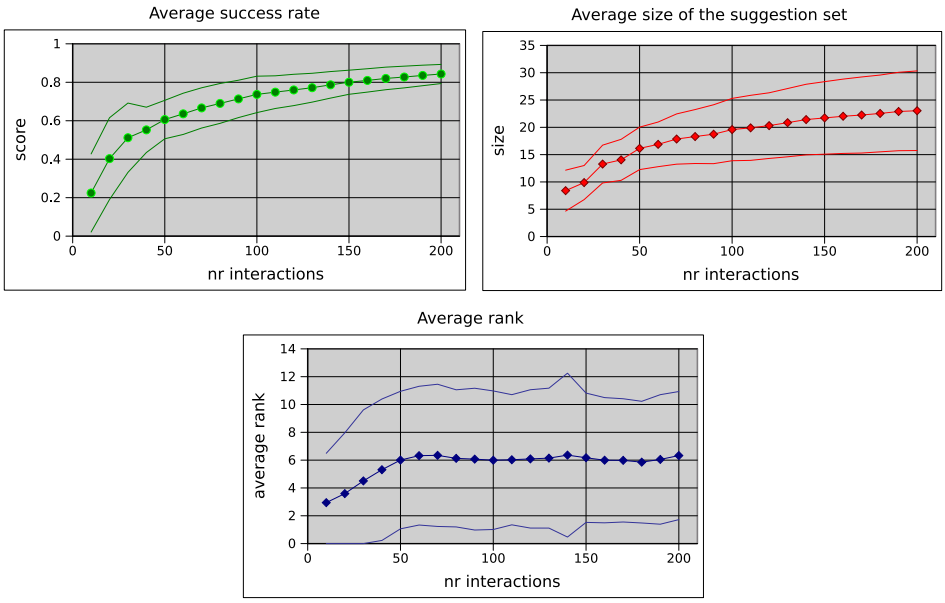

Figure 5.6: *Average size of the suggested set* Λ*, average success rate in finding* t_m *in it and average rank of* t_m *in* Λ

5.3.1 Specific methodology

The functionality experiments are run using three different ontologies, composed of 225, 626 and 1850 elements. These are generated varying the depths and the average numbers of children per node. Playing with these parameters it is possible to emulate flat lists without hierarchy, simple ontologies with shallow hierarchy, or more hierarchical structures. This allows to verify the performance of the predictor when dealing with different types of ontologies. See Figure 5.5 for an example of a generated taxonomy.

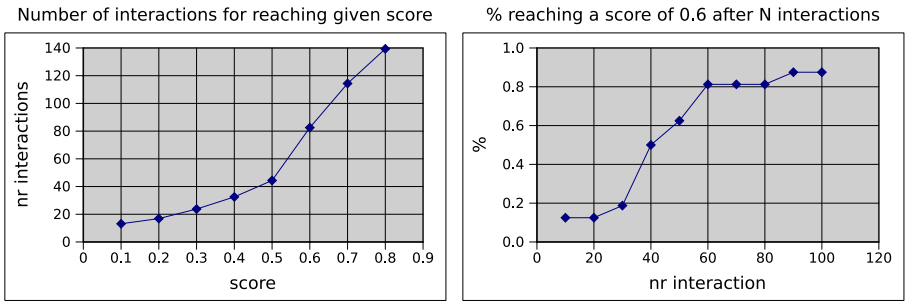

Figure 5.7: *Learning curve: average number of interactions needed to reach a given score, and probability of having a score of 0.6 after an increasing number of interactions.*

5.3.2 General Results

The performance of the predictor is measured by:

the average success rate, that is the average probability that t_m is in the suggested set Λ: $avg\,[P_Q(t_m \in \Lambda)]$ (where $avg\,[\cdot]$ is the average operator),

the average size of the suggested set Λ: $avg\,[|\Lambda|]$,

the average rank that the corresponding term t_m has in the probability distribution: $avg\,[rank\,(t_m, \mathbf{P}\,(Q_k))]$

Let us assume we know the exact probability distribution $\overline{\mathbf{P}}\,(Q_k\,|IM_{state}, M\,)$ of the terms for a random variable Q_k given the current context. As shown in Equation 3.5, given the list Ω of terms $t_j \in O$ ordered from the most likely to the least likely one the correct size n of Λ in order to obtain the desired success rate τ (i.e. the probability of finding t_m in Λ) is:

$$avg\,[p\,(t_m \in \Lambda)] = \tau = \textstyle\sum_1^n \overline{p}\,(t_j) \quad \text{where } t_j \in O$$

If the computed distribution $\mathbf{P}\,(Q_k\,|IM_{state}, M\,)$ is a good approximation of the exact distribution $\overline{\mathbf{P}}\,(Q_k\,|IM_{state}, M\,)$, then the average of $p(t_m \in \Lambda)$ should converge towards the average computed for $\overline{\mathbf{P}}\,(Q_k\,|IM_{state}, M\,)$ and therefore towards the threshold τ:

$$\lim_{nr\,interactions \to \infty} avg\,[p(t_m \in \Lambda)] = avg\,[\overline{p}(t_m \in \Lambda)] = \tau \tag{5.1}$$

If the success rate of the predictor remains lower than the threshold τ, independently of the number of interactions, then the computed distribution is different from the exact, but unknown, $\overline{\mathbf{P}}\,(Q_k\,|IM_{state}, M\,)$.

The size of the suggested set Λ will depend on the existence of relations between variables in the interaction and on the unknown distribution of terms in preference-based constraints, as we have seen in Section 3.4. These unknown distributions can change over time - if the phenomena are non-stationary - obviously decreasing the success rate. The lack of relations or flat distributions will cause large suggestion sets Λ, decreasing the usefulness of the predictor.

Another key issue to evaluate is the number of repeated interactions needed for the predictor to reach a stable behaviour. This number will be different for every type of interaction. What is necessary is to find its probability distribution, i.e. the probability that n interactions are enough to have a stable behaviour .

The results shown in Figure 5.6 were obtained averaging over the results of 12 different batches, generated combining 6 interaction models, 3 ontologies (225, 626 and 1850 elements) and different settings for the preference distributions (narrow and wide distributions for the preference-based constraints). All the batches were run with a threshold $\tau = 0.8$. The figure shows the average value of the size of the suggested set Λ and the average value of $p(t_m \in \Lambda)$, together with a band specifying the standard deviation of the measure. The limit in Formula 5.1 is verified, as the average score tends to stabilise, logarithmically, around τ (the standard deviation, showing fluctuations in success rate, decreases).

The average size remains small, independently of the size of the ontology, but its deviation tends to increase - albeit only logarithmically and remains well below 15% of the smaller ontology. The relatively large deviation reflects the fact that different batches have different relations between variable, and preference-based constraints have different distributions: therefore to obtain the same success rate the size of Λ may change meaningfully. However, the use of the filters on the assertions (described in Section 4.3) improved the results substantially: previous tests run on the same batches before the introduction of the filters returned the same average score, but a much higher average size (more than 150 elements instead of about 20).

The learning curve is, as stated, logarithmic: on average, most improvement (from 0 to nearly 70%) is obtained in the first 70-80 interactions, which is a small number of interactions in large peer-to-peer communities as those envisioned in the OpenKnowledge project. In the example scenario, the travel agency peer can be contacted by a thousand peers, all making similar requests, while the customer may need to contact several travel agencies before finding an appropriate accommodation.

Figure 5.7a shows the average number of interactions needed to reach different success rates, while Figure 5.7b shows the probability of having a success rate of 0.6 after an increasing number of interactions: the threshold $\tau = 0.8$ used in the experiments is reached on average after after 140 interactions, while 60 interactions are normally enough to reach a success rate of 0.6 on 80% of the experiments. Once in the stable region, the predictor will go on updating its representation, but the behaviour should change slowly or remain constant.

5.3.3 Analysing the results

We have discussed the average results shown in Figure 5.6 in the previous section: in the following subsections we will analyse how the predictor react in different situations. We first show how the probability distribution computed from the model is affected by different preference distributions over terms in messages. In subsection 5.3.5 we discuss how the various strategies that analyse the interactions and update the model contribute to the predictor performance. We then present how different preference distribution influences the performance, and how a non-stationary distribution (one that changes over time) affects the predictor and its analysis strategies.

5.3.4 Creating the model

The fundamental assumption is that if terms appear in messages in different runs of a interaction model according to an (unknown) probability distribution, then the system should be able to model this, updating the model interaction after interaction. Figure 5.8 shows how the predictor creates the probability distribution of a variable whose content is generated by preference functions with standard deviation $\sigma = 5$ and $\sigma = 25$, after 30, 60 and 120 interactions. It is possible to see that the model gets closer and closer to the half-normal distribution with which the terms are generated, and that the model moves more slowly towards the exact distribution when the terms in the predicted variable are distributed with a wider distribution ($\sigma = 25$).

5.3.5 Contributions of the strategies

In Section 3.4 we made four assumptions about the terms in the interactions, that we transformed into four types of assertions, two based on the frequency of terms and two based on their ontological relations, as we showed in Section 4.3. We need to evaluate

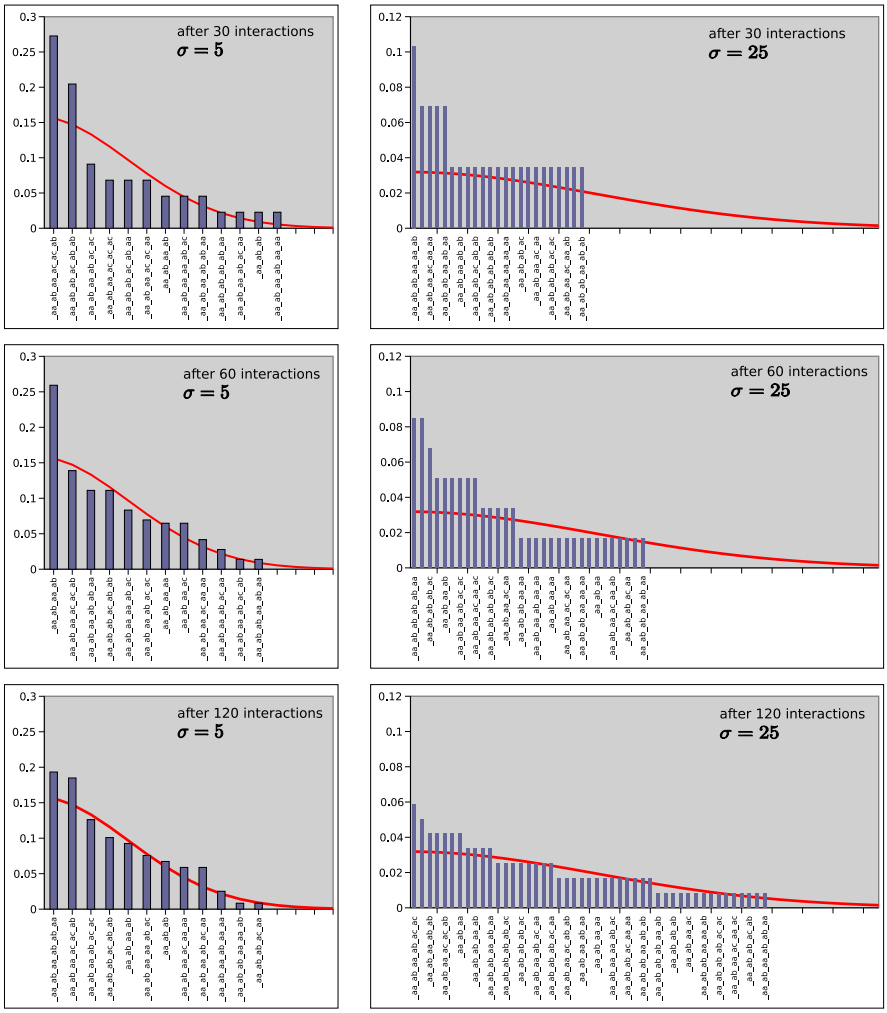

Figure 5.8: *How the model improves after 30, 60, and 120 interactions with* $\sigma = 5$ *and* $\sigma = 25$

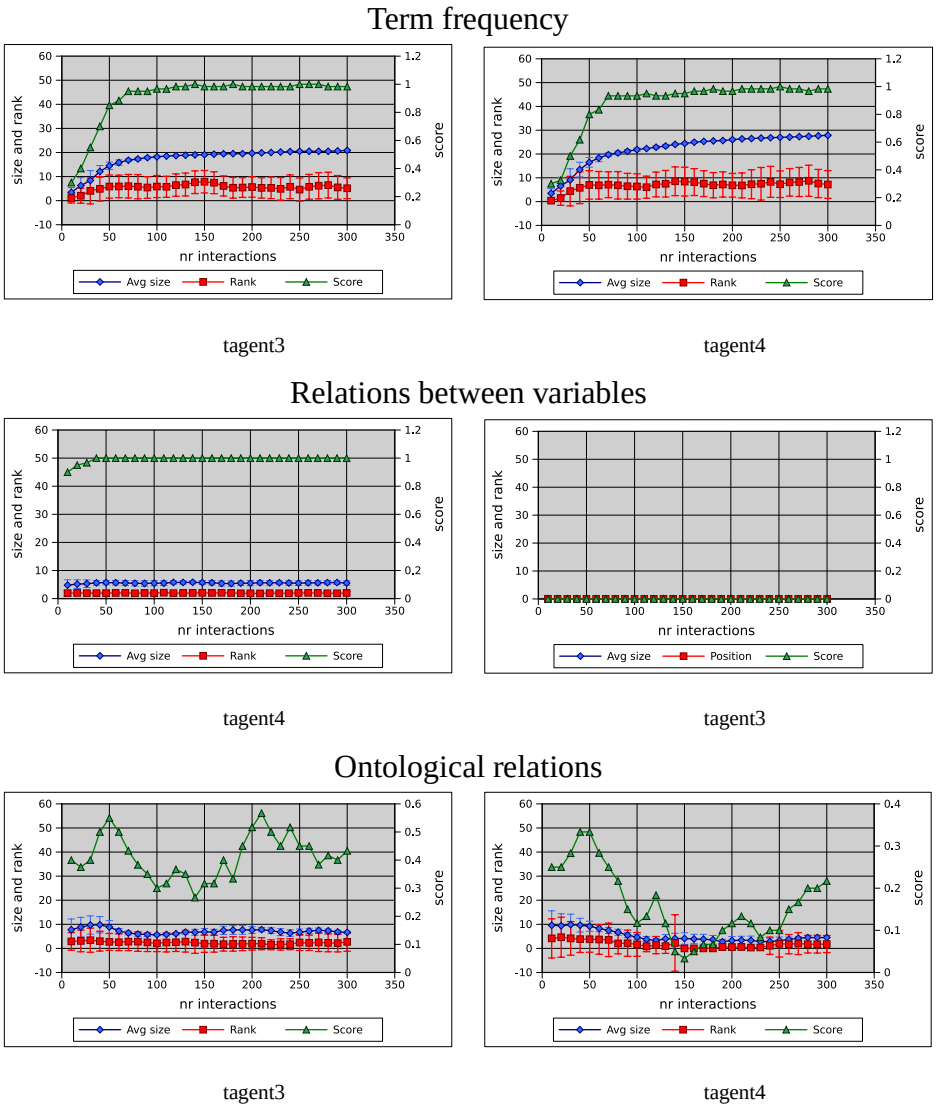

Figure 5.9: *Contribution of different types of assertions. Tagent3 predicts a variable whose content is related to another known variable, while tagent4 predicts a variable whose content depends only on a preference distribution.*

how these assertions contribute to create the model, when they help and when they do not add useful information.

Figure 5.9 shows the different contributions of the strategies to the performance of the predictor: each batch of experiments was run using a single strategy for generating assertions and averaging the results obtained varying distributions and relations between the terms in messages. In the graphs `tagent4` needs to predict a purely preference based variable, while `tagent3` needs to predict a variable that has a relation with another variable.

The most consistent type of assertions is term frequency when the distribution does not change over time, as it generates a set that contains the correct term rather quickly.

Assertions about relations between variables are successful when there are relations to find, and reach a high score very quickly. The size of the suggestions depends on the peer ontology (large and shallow ontologies behave worse than thin and deep ones). However, these assertions are not created - or are discarded by the thresholding mechanism when spurious ones are created - when there is no relation, and therefore cannot help in these cases, as shown by the `tagent4` graphs in Figure 5.9.

The experiments using only the conditional frequency showed no useful results: the success rate was always 0. One of the problems that arose in analysing these results was the sparseness of the results: there were too many assertions, each capturing one case with very low frequency. Conditional frequency only makes sense when the vocabulary used in messages is small, otherwise it requires a vast number of interactions to provide useful information. For example, if the content a of a first message in an interaction is taken with a uniform probability from a set of 20 terms, and the content b of the following message is taken from a set of 200 terms, where there are 10 possible different terms for each term in a, then after 200 interaction there might be 200 assertions, each stating one particular case. Another possible issue is the distance considered between the variables: in the experiments a distance of 1 was used, but it might be that meaningful relations are between variables slightly further apart, as shown by the ontological relations described before. An interesting extension could be to store assertions about the posterior probabilities of all the variables in an interaction model, and then use only those that present higher frequencies. Such a strategy should generate several assertions about unrelated variables, each with very low frequency, and fewer assertions with higher frequency about related variables.

Assertions about ontological relations between the terms in the messages and the peer's ontology tend to provide a rather unstable contribution: the score of the predictor

fluctuates between 0.2 and 0.6 for `tagent3` and between 0.05 and 0.35 for `tagent4` when it uses only this kind of assertion. The only relation that is verified and stored is the subclass relation: when a term t_i appears in the message fed back from the mapping oracle, its superclass $t_s \sqsubseteq t_i$ is found in the peer's ontology and the assertion about the subclasses of t_s is stored or updated. However, it may not be the case that all the sibling terms of t_i are equally likely to appear, while the assertion makes this assumption.

5.3.6 Case analysis

Section 5.3.2 presented the general behaviour of the predictor, and the Figures 5.6 and 5.7 explained in the section are obtained averaging many runs of different types of experiments. Section 5.3.5 evaluated how the different strategies used to analyse the runs contribute to the overall results of the predictor. In this section we will evaluate the performance of the predictor in different scenarios. In particular we focus on how the performance degrades when the distribution of terms, representing the preferences of the community of users, varies in breadth, and when it varies over time.

Wide vs narrow preference distributions

The content of messages in interactions can exhibit varying level of randomness. The content of a message may alternate among only a few terms, with one or two terms more frequent than the others, or it can be any term from a wider range of possible ones where all are equally likely.

In my tests, this is simulated varying the width, given by the standard deviation σ, of the Gaussian distribution used to generate the content of the messages. Figure 5.10 shows the effects on the average size, the score and the rank of the correct term for three distributions of increasing width, with σ equal to 5, 10, 15, 25. The interaction model used is a variation of the standard one: a message, whose content is randomly chosen according to the above distributions, is sent by `tagent3` to `tagent4`. The recipient replies with a term ontologically related to the received term (for example it can be a subclass or a property). Therefore, `tagent4` has to predict a term that depends only on an external distribution, while `tagent3` has to predict a term that depends on a term he has chosen.

When the content of the message is ontologically related to another known term, as in `tagent3` case, the performance is not meaningfully influenced by the changes in

the distribution of the known term. On the other hand, when the content depends only on an external, unknown distribution, as in `tagent4`, the performance is heavily influenced. The average size of the suggested Λ increases with σ: after 200 interactions, the average size is around 10 for $\sigma = 5$ and reaches nearly 50 for $\sigma = 25$. The score always converges towards 1, but the slope steepness decreases with σ and oscillations increase with it. The average rank of the correct term increases, although less than the average size, but variation in rank increases notably.

Non-stationary distributions

As discussed in Section 3.4, results returned by preference-based constraints follow a distribution that reflects the contingent preferences or needs of the user community. As we have seen in Section 5.3.5, a variable whose value depends exclusively on community preferences is modelled mainly by assertions describing the prior frequencies of terms. If the preference distribution is not stationary and changes over time the assertions built after a number of interactions may not model the variable distribution correctly in new interactions. In particular, variables whose values are predicted only by assertions based on term frequencies will be affected most, while variables depending on some rules or functions should be more robust when preferences change, as the assertions model the ontological relation between the term in the variable and the value of other variables that can be assumed to be independent from the distribution of terms.

To test the behaviour of the predictor when dealing with non stationary preferences, we run two batches of experiments, both using the recursive interaction model in Figure 5.11. In the interaction model, the agent performing role `r9a1` sends a message about X, where the value of X is chosen from a preference distribution. The agent performing role `r9b1` receives the message, finds a list of elements related to X and starts sending them back to the first agent. The first agent can either accept the term, or ask for more.

The first batch is used as a baseline: it is composed of three experiments, each of 300 interactions and the preference distribution for variable X is stationary. The second batch is composed of three experiments, each consisting of 300 interactions. In these three experiments the preference distribution for variable X is *non-stationary*, changing every 100 interactions. The performance measures in the two batches are averaged.

The results are shown in Figure 5.12. The prediction for variable Y that depends

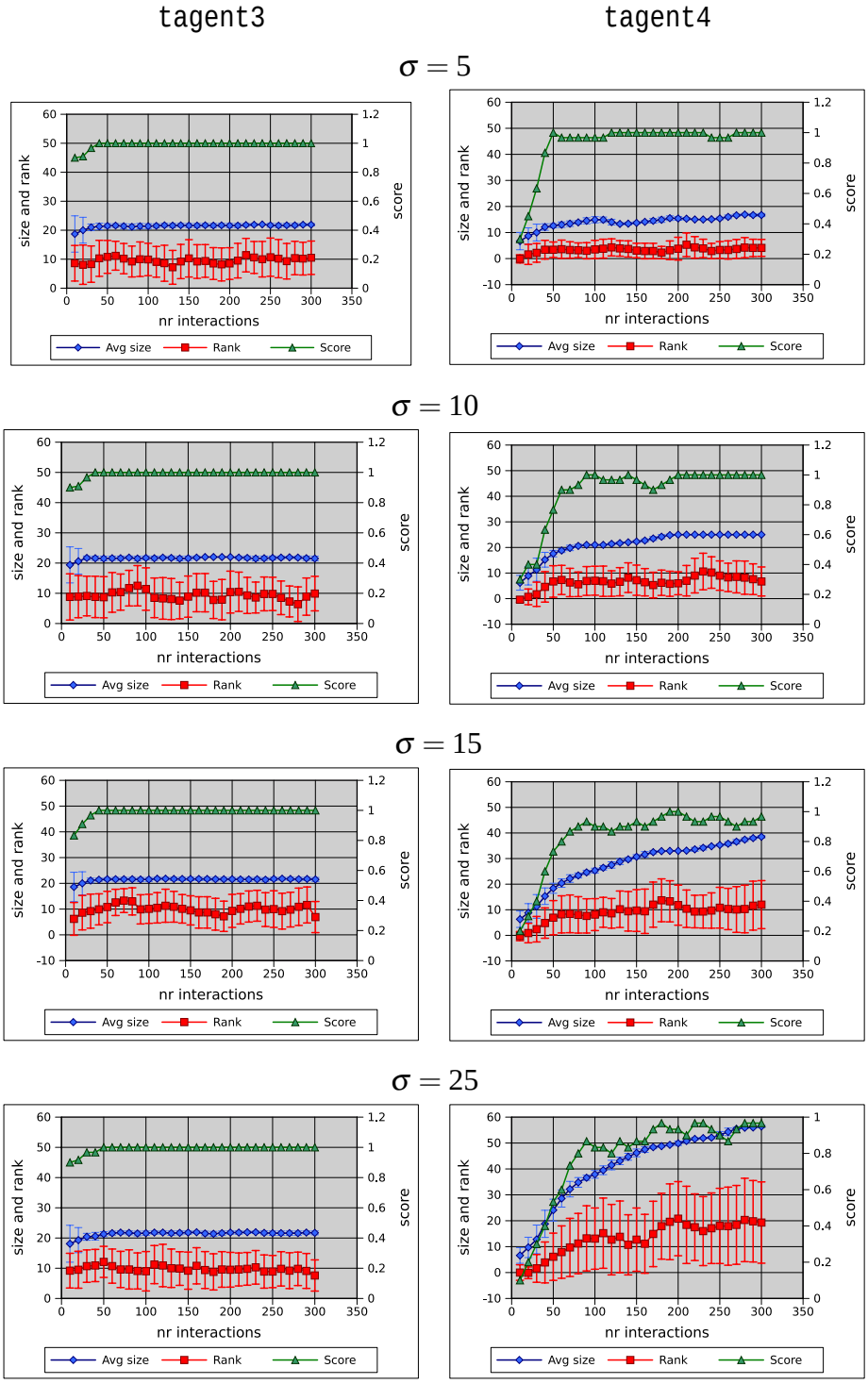

Figure 5.10: *Effect of different preference distributions.* `tagent3` *predicts a variable whose content depends on a variable with different preference distributions, while* `tagent4` *predicts a variable whose content depends only on the different preference distributions.*

$$
\begin{array}{l}
\mathtt{a(r9a1(ID2),ID1)} :: \\
\mathtt{m_1(X)} \Rightarrow \mathtt{a(r9b1,ID2)} \leftarrow \mathtt{kp9\_1(X)\ then} \\
\mathtt{a(r9a2(X),ID1)} \\
\\
\mathtt{a(r9a2(X),ID1)} :: \\
\left(\begin{array}{l}
\mathtt{m_2(Y)} \Leftarrow \mathtt{a(r9b2,ID2)\ then} \\
\mathtt{m_3} \Rightarrow \mathtt{a(r9b2,ID2)} \leftarrow \mathtt{kp9\_2(X,Y)} \\
\mathtt{or} \\
\left(\begin{array}{l}
\mathtt{m_4} \Rightarrow \mathtt{a(r9b2,ID2)\ then} \\
\mathtt{a(r9a2(X),ID1)}
\end{array} \right)
\end{array} \right) \\
\mathtt{or} \\
\mathtt{m_5} \Leftarrow \mathtt{a(r9b2,ID2)} \\
\mathtt{a(r9b1,ID2)} :: \\
\mathtt{m_1(X)} \Leftarrow \mathtt{a(r0a1,ID1)\ then} \\
\mathtt{a(r9b2(ID1,Lst),ID2)} \leftarrow \mathtt{kp9\_3(X,Lst)} \\
\\
\mathtt{a(r9b2,(ID1,Lst),ID2)} :: \\
\mathtt{m_2(Y)} \Rightarrow \mathtt{a(r9a2,ID1)} \leftarrow \mathtt{Lst} = [\mathtt{T}|\mathtt{Tail}]\ \mathtt{then} \\
\left(\begin{array}{l}
\mathtt{m_3} \Leftarrow \mathtt{a(r9a2,ID1)} \\
\mathtt{or} \\
\left(\begin{array}{l}
\mathtt{m_4} \Leftarrow \mathtt{a(r9a2,ID1)\ then} \\
\mathtt{a(r9b2,(ID1,Tail),ID2)}
\end{array} \right)
\end{array} \right) \\
\mathtt{or} \\
\mathtt{m_5} \Rightarrow \mathtt{a(r9a2,ID1)}
\end{array}
$$

Figure 5.11: *Recursive test interaction model. The peer taking role* r9a1 *starts the interaction solving constraint* k9_1 *in order to find a value for* X. *It first sends the value to the peer in role* r9b1 *and then takes the recursive role* r9a2. *The peer in role* r9b1 *obtains a list of options, stored in* Lst, *from the received value* X *by solving the constraint* k9_3. *Then it takes the recursive role* r9b2 *and sends the first option in* Lst *with message* $\mathtt{m_2(Y)}$. *If there are no options, it sends message* $\mathtt{m_5}$. *The initiator peer, now in role* r9a2, *receives the message containing the option, evaluate it solving constraint* k9_2 *and either accepts it, sending message* $\mathtt{m_3}$ *or rejects it, sending message* $\mathtt{m_4}$. *If there were no options, it would have received message* $\mathtt{m_5}$, *and it would have terminated the interaction. The peer in role* r9b2 *waits for one of the two messages* $\mathtt{m_3}$*or* $\mathtt{m_4}$*: if the acceptance arrives, it terminates the interaction, otherwise recurses passing the remaining options.*

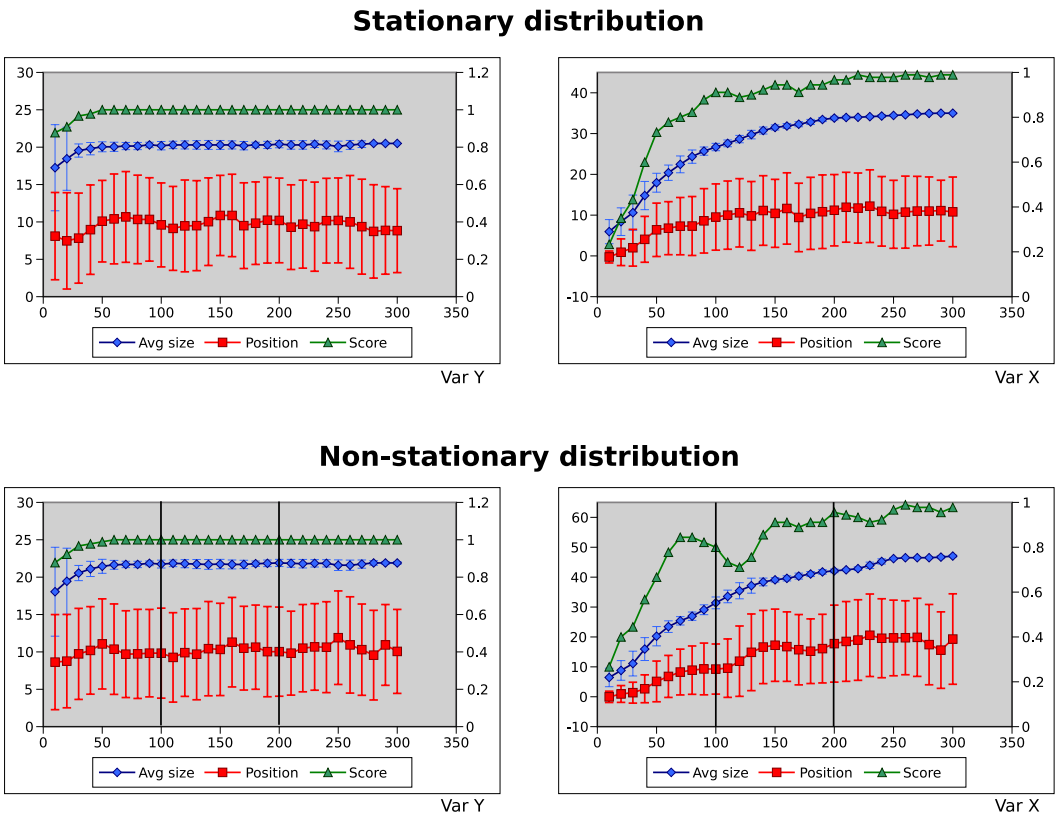

Figure 5.12: *Predictor behaviour when distribution changes over time*

on the value of the first variable X, quickly stabilises in both cases, and is not affected by changes in the distribution of X: the ontological rule found by the predictor is independent of the distribution of the first variable. Performance regarding the prediction of variable X, on the other hand, depends on whether the X is stationary or not. If variable X is stationary, average size and score grow logarithmically, and the position of the correct term t_m increases during the first 50 interactions and remains more or less constant in the remaining ones. If variable Y is non-stationary, then the score grows as in the stationary case up to 85% until the distribution is changed, where it suddenly decreases to 70%. The average size grows more rapidly after the change. The score returns to its previous value after 100 interactions. The second change of distribution, after 200 interactions, has a much lower impact, as it is only an increase of the spread of the distribution. For variable X, when the distribution changes, the size Λ after 300 interactions is much bigger.

5.4 Verifying Usefulness

The main goal of the predictor is to provide a set of likely terms to an Ontology Matcher, so that it can focus on them and find the correct correspondence for a foreign term using fewer computational resources. To evaluate the contribution of the predictor, we tested the results of the predictions on a real ontology matcher, using peers with different ontologies.

5.4.1 Specific methodology

Two different ontologies are used. The first ontology is a generated tree: labels in nodes are composed of a random number of words, selected from 9000 words extracted from part of the Brown Corpus, and the number of children for each node follows a Gaussian distribution, with average 4, deviation 4. The maximum depth is 4 and the overall size is 986 nodes. The second ontology is obtained from the first, applying the changes described in Table 5.1. Its overall size is 1000 nodes.

As stated in Section 5.2, one of the reason for using generated ontology instead of existing ones is that they allow a wider coverage of variations in their structure. In this specific case, the matching between ontology is evaluated: the use of existing ontologies is possible only if correspondences between them exist as well, further reducing the possible variations that can be explored.

The matcher used is described in the next section: applying it on the entire ontologies, without the involvement of the predictor, yields a recall rate of 0.7 and a precision of 0.85.

The Ontology Matcher

The aim of the experiment is to verify how the predictor can improve the performance of a generic ontology matcher, and therefore a relatively simple matcher was selected. The matcher `pyontomap` [4] used in the experiments is composite matcher: it employs a set of standard elementary matchers (syntactic, structural and semantic) and combines their results using a Dempster-Shafer [68] based algorithm. While in the Bayesian approach probabilities are assigned to single entities, in Dempster-Shafer the mass is distributed on *sets* of propositions. The mass distribution is a function $m(\cdot)$ that distributes a mass in the interval [0,1] to each element of the power set 2^{Θ} of the set of propositions $\Theta = \{\theta_1, \theta_2, \ldots, \theta_n\}$ called the *frame of discernment*. The total mass

Tree alteration:
For each node apply:

- label replacement, with probability 0.01
- syntactic label alteration, with probability 0.2 (letters dropped, added, changed)
- word addition or removal in labels, with probability 0.15
- word replacement in labels, with probability 0.4, choosing from:
  - synonyms, hyponyms, hypernyms (extracted from WordNet 3.0, using all the possible parts of the speech of the word)
  - related words (extracted from the Moby thesaurus)
- node deletion (the number of nodes to remove is computed using a Gaussian distribution with average 0 and standard deviation 0.9)
- new child addition, with probability 0.25
- children shuffling, with probability 0.4

Table 5.1: *Tree creation and alteration process. The probabilities of the alteration operations have been chosen by trial and error in order to obtain reasonably altered trees, without having completely unrelated trees.*

distributed is 1 and the *closed world assumption* is generally made: the frame Θ contains the true hypothesis. This is expressed assigning mass 0 to the empty set $\emptyset$, called contradiction. The mass $m(\Theta)$ assigned to the frame represents the mass that cannot be assigned to any particular subset of Θ. Different mass distributions can be combined using *Dempster's rule of combination* that computes the probability mass assigned to $C \subseteq \Theta$ given $A \subseteq \Theta$ and $B \subseteq \Theta$, where A is supported by m_1 and B is supported by m_2:

$$m(C) = \frac{\sum_{A \cap B = C} m_1(A) m_2(B)}{1 - \sum_{A \cap B \neq \emptyset} m_1(A) m_2(B)} \tag{5.2}$$

Once the masses have been distributed and combined, it is necessary to extract the most likely entity from the mass distribution. Dempster-Shafer makes it possible to compute

the *belief* about a set $A \subseteq \Theta$ of propositions, as the sum of all the basic masses that support its constituents:

$$Bel(A) = \sum_{B \cap A} m(B)$$

It also provides the formula for computing *plausibility* of the set A, that is the measure of the extent to which A might be true:

$$Pl(A) = 1 - Bel(\overline{A}) = \sum_{B \cap A \neq \emptyset} m(B)$$

In the matching process, a term from ontology O_1 is compared, using all the matchers, with all the terms from ontology listed in set $T \subseteq O_2$. The set T represents the frame of discernment. The results of the comparisons performed by an elementary matcher are split into sets: each set contains terms that are equally likely to be the exact alignment, and it is given a mass representing the likelihood that the exact match is contained. The sets generated by the different matchers are combined using Dempster's formula, and then belief is computed for each term.

For example, given the term *bed* in O_1 and the elementary matcher Edit-Distance, the terms *bid* and *bad* from O_2 are equally likely to be the correct correspondence (they both have a distance of 0.33), and are put in the same set. Sets containing terms with distance between 0 and 0.2 are given weight 0.5, those with terms having distance between 0.2 and 0.3 are given 0.3, and finally those with terms having distance between 0.3 and 0.5 are given 0.2. Terms with greater distance are discarded, giving them mass 0. The mass that cannot be given to any term is assigned to the set $T \subseteq O_2$ (that forms the frame of discernment Θ), and represents the "ignorance" of the matcher: a matcher unable to find any similarity between a foreign term and all the terms in peer ontology will give all of its mass to the frame of discernment. Continuing with the previous example, if two matchers return:

- $m_1(\{bad, bid\}) = 0.33$, $m_1(\{bed\}) = 0.5$, $m_1(\Theta) = 0.17$
- $m_2(\{but, bid, bar\}) = 0.1$, $m_2(\{bed\}) = 0.6$, $m_2(\Theta) = 0.3$

where $m_1(\Theta)$ and $m_2(\Theta)$ are the masses given by the matchers to the set $T \subseteq O$, and represent the masses that cannot be assigned to any particular set. The combined mass distribution will be:

$infix(t_1, t_2)$ checks if t_1 is contained in t_2 or t_2 in t_1

$postfix(t_1, t_2)$ checks if t_1 ends with t_2 or the otherway around

$prefix(t_1, t_2)$ checks if t_1 starts with t_2 or the otherway around

$soundex(t_1, t_2)$ checks for soundex similarity between t_1 and t_2

$editdistance(t_1, t_2)$ checks for the edit distance (number of string changes - addition, deletion and modification of characters - needed to reach one string from another) between t_1 and t_2

$initsmatch(t_1, t_2)$ checks if the initials of t_1 correspond to t_2 or the other way around

$parents(t_1, t_2, O_1, O_2)$ checks the edit distance of the parents of t_1 and t_2

$children(t_1, t_2, O_1, O_2)$ checks the edit distance of the children of t_1 and t_2

$siblings(t_1, t_2, O_1, O_2)$ checks the edit distance of the siblings of t_1 and t_2

Table 5.2: *Matchers used in* `pyontomap`

$m_1 \otimes m_2$	$m_1\left(\left\{\begin{array}{c} bad \\ bid \end{array}\right\}\right) = 0.33$	$m_1(\{bed\}) = 0.5$	$m_1(\Theta) = 0.17$
$m_2\left(\left\{\begin{array}{c} but \\ bid \\ bar \end{array}\right\}\right) = 0.1$	$m_{1\otimes 2}(\{bid\}) = 0.033$	$m_{1\otimes 2}(\emptyset) = 0.05$	$m_2\left(\left\{\begin{array}{c} but \\ bid \\ bar \end{array}\right\}\right) = 0.017$
$m_2(\{bed\}) = 0.6$	$m_{1\otimes 2}(\emptyset) = 0.2$	$m_{1\otimes 2}(\{bed\}) = 0.3$	$m_{1\otimes 2}(\{bed\}) = 0.102$
$m_2(\Theta) = 0.3$	$m_{1\otimes 2}(\{bid\}) = 0.099$	$m_{1\otimes 2}\left(\left\{\begin{array}{c} bad \\ bed \end{array}\right\}\right) = 0.15$	$m_{1\otimes 2}(\Theta) = 0.051$

The beliefs about the alignments are:

$$Bel(\{bed\}) = 0.15 + 0.3 + 0.102 = 0.552$$
$$Bel(\{bid\}) = 0.033 + 0.099 = 0.132$$
$$\vdots$$

The matcher was configured to use the matchers in Table 5.2.

Using the predictor

As we have seen in Section 3.5, the probability distribution $\mathbf{P}(Q_k | IM_{state}, M)$ computed by the predictor can be used:

1. to extract a subset Λ of terms from the peer's ontology to be compared with

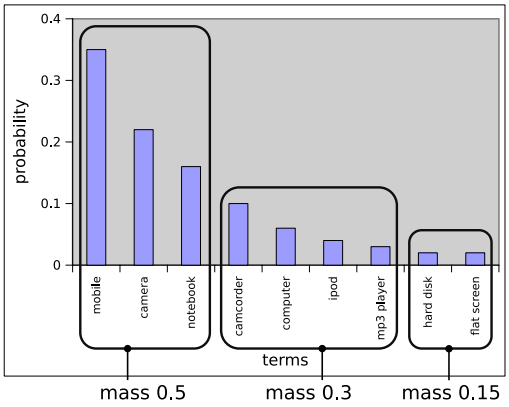

Figure 5.13: *Splitting the probability distribution into sets*

the term in the message, reducing the resources required for matching (setting $T = \Lambda$), and

2. as results of an additional matcher, able to exploit the additional information available in the context of the interaction.

In the first case, if nothing is found by the matcher in the suggestion set Λ (that is, there is no term with belief higher than a given threshold), it is possible either to consider that no possible match exists (*no reattempt* policy), or to extend the comparisons to the rest of the ontology, posing the set $T = O_2 \backslash \Lambda$ (*reattempt* policy).

If $\mathbf{P}(Q_k | IM_{state}, M)$ is used as the result of an additional matcher, the distribution is split into sets of terms equally likely to be the exact match and a mass is assigned to each set, as shown in Figure 5.13. The thresholds for splitting the probability distribution into sets and the masses assigned to the sets were obtained empirically.

Running the experiments

Each performed experiment consists of running 400 interactions: first 200 interactions are run between two agents with the same ontology with the aim of creating a first approximation of the statistical model, then 200 further interactions are run replacing one agent with another that uses a different ontology. The predictor is not aware that the ontology is shared in the first set of runs, and works as if it had to predict and match different ontologies. As described above, one ontology is generated, while the second is a variation of the first ontology, obtained applying the changes described in Table 5.1.

Three different types of experiments were executed: one without the use of the predictor, as a baseline, and two using the predictor, the first using the *no reattempt*

policy (no match exists if nothing is found in Λ), and the second using the *reattempt* policy (extending the comparisons to the remaining ontology if nothing is found in Λ). Each type of experiments was run 3 times to average the results.

The experiments were run on a dual core laptop with two 1.83 T5600 CPUs and 1Gb of RAM.

5.4.2 Results

We have seen in Section 3.2 that the performance of an ontology matcher is usually measured by its *precision* and *recall*. Given that M_{found} is the set of correspondences found by the mapping system and $M_{correct}$ is the set of correct correspondences:

precision is the ratio between the number of correct correspondences among those found and the total number of correspondences found:
$Precision = \frac{|M_{found} \cap M_{correct}|}{|M_{found}|}$

recall is the ratio between the number of correspondences found and the total number of possible ones:
$Recall = \frac{|M_{found} \cap M_{correct}|}{|M_{correct}|}$

The average size and the average success rate of the predictor influence the performance of the matcher when the probability distribution is used only to generate the suggested set Λ. If the *no reattempt* policy is used, then a low success rate will surely lower the recall, and possibly the precision. A low success rate means that the corresponding term t_m is often not in the suggested set Λ: many possible correspondences will be missed by the matcher that uses only the terms in Λ for comparison with w_j, reducing the set M_{found}. Precision is lowered as well, but by a different mechanism. A set Λ not containing t_m may contain another term t_{wm} that is considered to correspond to w_j "well enough" by the matcher: the belief in its correspondence is lower than what would be computed for t_m if it was in Λ, but it might still be higher than the threshold, and because there are no competitors, it is chosen as the best correspondence, lowering precision.

If the *reattempt* policy is used, then a low success rate will lower the precision for the same reason as above, but recall will be affected less: if nothing is found in Λ, then the remaining terms in the ontology are compared with w_j, increasing the likelihood of finding the correspondence.

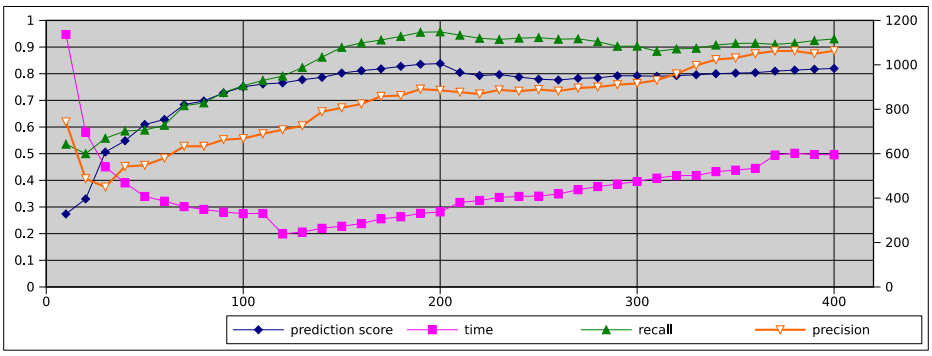

Figure 5.14: *Matching results when predictor is used. Finding no correspondences in the suggestions set Λ is considered equivalent as finding no correspondence at all (no reattempt policy)*

If the probability distribution is used as a matcher, then it influences directly the belief computed for the terms. If the probability assigned to t_m in $\mathbf{P}(Q_k|IM_{state}, M)$ is consistently low, then a low mass will be assigned to the term: this influences the belief in t_m, as we have seen in Section 2.6.

Figure 5.14 shows the results of running the experiment with the predictor, with the *no reattempt* policy. What the graph shows is that the time required for matching drops immediately, keeps decreasing for a while and then slowly increases. This trend reflects the fact (mentioned in Section 5.3.2) that the average size of Λ is low initially and increases with every interaction: the number of comparisons increases proportionally with the size of Λ. Precision and recall are small initially, and increase following the success rate of the predictor.

Figure 5.15 shows the results of running the experiment with the predictor, using the *reattempt* policy. Time decreases while the predictor improves its success rate, and stabilises when the predictor success stabilises. Recall and precision decrease initially and then increase converging towards respectively 1 and 0.9.

If, as described in the second use of the predictor, the probability distribution $\mathbf{P}(Q_k|IM_{state}, M)$ is used as an additional matcher, assigning a low probability to the correct term and high probability to the wrong ones sways the mass distribution computed combining the mass distributions provided by the other ontology matchers. The probability distribution $\mathbf{P}(Q_k|IM_{state}, M)$ is split into sets containing terms with similar likelihood. If the probability distribution is not correct, the wrong terms will receive more mass than the correct one, and the combination of masses computed using Dempster's rule will tend to sway mass towards the wrong terms. This is particularly

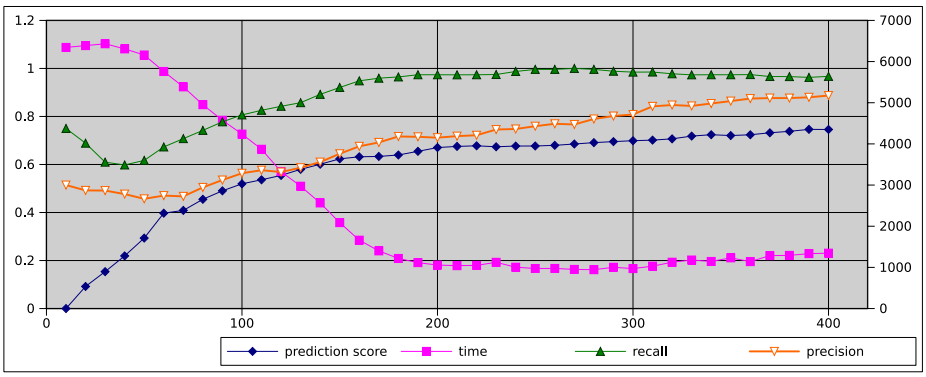

Figure 5.15: *Matching results when predictor is used. When no correspondence is found in the suggestions set Λ, the matcher is used to compare the remaining terms in the whole ontology (reattempt policy).*

problematic when the ontology matchers can assign only little mass, and are forced to assign most of their masses to the frame of discernment because of lack of information about the relations between the terms to match. The mass assigned using the probability distribution will override the mass assigned by the other matchers.

Initially, the predictor is bound to have the wrong distribution, as the results provided earlier show: it takes at least 80-90 runs to obtain a consistent success rate of 60%. To compensate for this, the mass that can be assigned by the predictor is initially low, and increases over time, following a logarithmic curve similar to the learning curve obtained empirically and shown in Figure 5.7. During the first runs of an interaction, the predictor splits a small amount of mass between the sets of equally likely terms, and assigns the remaining mass to the frame of discernment. As the interaction is repeated, the statistical model gets better (on average) and the mass that the predictor can split between the sets increases.

5.4.2.1 Comparing performance

Time

Figure 5.16 compares directly the computation times for the three cases (no predictor used, predictor used with *reattempt* policy and predictor used with *no reattempt* policy). When no predictor is used, the number of comparisons remains constant over 400 interactions, and therefore time remains constant around 10000ms: as we said earlier, matching is always performed and comparing terms from the same ontology is no quicker than comparing from two different ones. Fluctuations are due to different CPU loads over time.

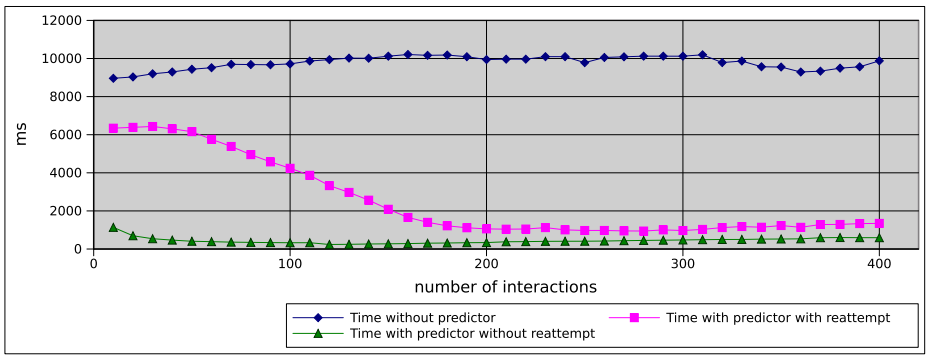

Figure 5.16: *Matching time when predictor is not used, is used with reattempt and without reattempt.*

The use of the predictor reduces time complexity remarkably. When the *no reattempt* policy is used, matching time starts at a low value of 1200ms (it compares always only the terms in Λ), decreases further to 350ms and remains low, increasing only slightly to 600ms with the increasing size of Λ, as we have seen before. When the *reattempt* policy is used, the matching time starts at 6400ms because the initial success rate is only 0.4 and therefore in 60% of the cases the comparisons are done with the whole ontology. As the success rate increases, time decreases and stabilises at around 1000-1200ms, a level twice the one obtained using the *no reattempt* policy but nearly 10 times lower than that needed by the baseline solution. The average success rate of the predictor, as we have seen before, is around 0.8: this means that in up to 20% of the cases nothing is found, and comparisons have to be performed with the remaining terms in the ontology. As pointed out in the introduction of this section, if the exact correspondence t_m is not in the suggested set Λ the wrong correspondence can be found in it, reducing precision but also computation time as a side effect (no further comparisons with the remaining terms in the ontology are required).

Precision

Figure 5.17 compares precision across the three experiments. In the baseline solution, where no predictor is used, precision fluctuates around 0.9 after the first 200 interactions in which the same ontology is used by both peers.

In the experiments with the predictor, precision starts remarkably lower than the baseline and then linearly converges towards the baseline. There is no evident difference between the two policies (*no reattempt* and *reattempt*) when the predictor is used.

We have seen in Section 5.3.2 that the success rate starts at a low value, and there-

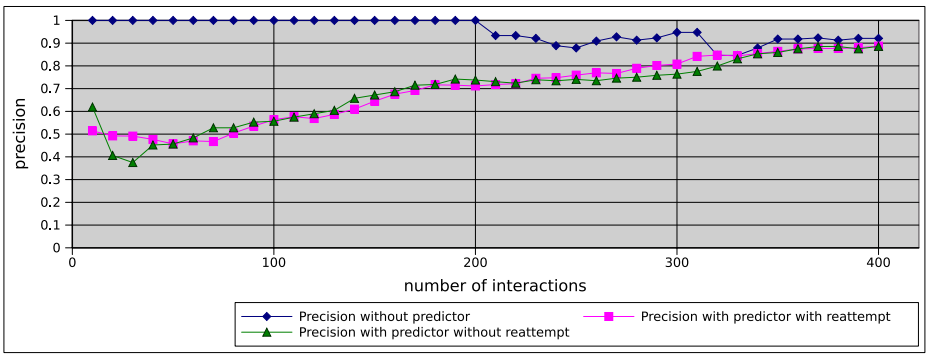

Figure 5.17: *Matching precision when predictor is not used, when used with reattempt strategy and without reattempt strategy.*

fore initially the suggested set Λ often does not contain the correct correspondence: we have explained above that the matcher may find a term t_{wm} in Λ whose belief is higher than the threshold, and it is wrongly chosen as the correct alignment, lowering the precision.

We have also illustrated in Subsection 5.4.2 that if the probability distribution $\mathbf{P}(Q_k|IM_{state}, M)$ is used as an additional matcher, it may sway the combined mass when it ranks as unlikely the correct term, especially when the other matchers can distribute little or no mass.

Recall

Figure 5.18 compares the recall trend in the three experiments. In the baseline, where no predictor is used, recall stabilises around 0.95, after the first 200 interactions in which the same ontology is used. When the predictor is used, recall starts lower, decreases and then converges towards the same value as the baseline. Using the *reattempt* policy, recall overtakes the baseline, remaining constantly higher. Using the *no reattempt* policy, recall starts lower than with the *reattempt* policy and remains lower (15-20%) than the baseline for most of the experiment, getting closer only towards the end of the experiment.

Compared to the baseline, precision is sometimes improved by the additional information, but, as we have seen above, the failure to include the exact correspondence in the suggestion set Λ can sway the matcher towards selecting the wrong term. Recall, on the other hand, is improved by the additional information provided by the predictor.

The fluctuations in both precision and recall depend also on the terms w_j randomly chosen for the messages: within the 10 interaction interval there might be different numbers of terms that the matchers cannot map correctly.

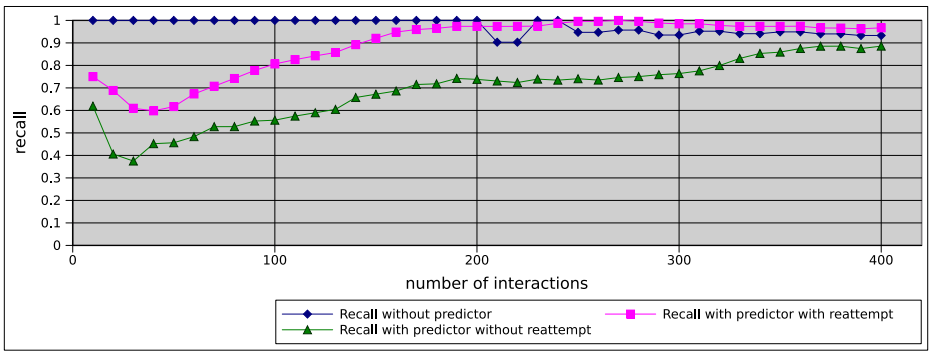

Figure 5.18: *Matching recall when predictor is not used, is used with reattempt and without reattempt.*

5.5 Summary

In this chapter we have first evaluated the performance of the content predictor presented in Chapter 4 independently of its use and then to assess the benefits of using it with an Ontology Matching system.

The evaluation was performed by simulation: the tested interactions represent patterns of common interactions, and the peers respond to constraints either using a probability distribution over the possible values (to reflect the preferences of a community) or according to some specific function. The peers were given generated ontologies. This allows the evaluation of the predictor when used with ontologies varying along different dimensions, specified during generation.

As said in Chapter 4, the predictor computes a probability distribution for a particular variable in a received message, using the context information available from the current and the past interaction runs. The probability distribution can be used to select the most likely terms (those whose cumulative probability is higher than a given threshold), and as a synthetised contextual information that can be exploited by a matching algorithm.

When evaluating the ability of the predictor in guessing the correct content of the exchanged messages, no matching was involved: the peers interact sharing the same ontology. The aim of this set of experiments was to evaluate how different interaction scenarios could be handled by the predictor. The scenarios were simulated varying the preference distributions used to select the terms to introduce in variables: narrow, wide and time-varying distributions were used. The measures of performance considered are the size of the suggested set Λ of likely terms (see Section 3.5), the probability that the set contains the exact term in the message, and the rank of the term in the set.

The usefulness of the predictor has been evaluated feeding the results into a matcher that must map the foreign terms in the messages to local terms. The performance is compared with a baseline case in which the predictor is not used. The computational time required by the matcher to find the correspondence, keeping the precision and recall constant, is reduced by a factor of 8 to 10. On the other hand, when a new interaction is used recall and in particular precision start low, and increase at the same rate of the success rate of the predictor. However after enough interactions, precision reaches the same level of the baseline and recall reaches a slightly higher level.

Chapter 6

Related Work

6.1 Introduction

The work presented in this thesis pulls together different technologies: it does so not with the aim of improving any of them, but with the aim of showing how they can be brought together in a novel way in order to improve their overall performance. In particular, it exploits a model of agent coordination for analysing the interactions between agents and it feeds the results of the analyses to an ontology matcher. The results fed to the matcher are predictions of the likely content of the exchanged messages between the agents. The predictions, extracted from analysing dialogues, are obtained using statistical methods partially inspired by Natural Language Processing techniques.

Different agent coordination approaches are available, but not all of them can be successfully used with the framework described in the thesis: the predictor needs a coordination model that considers interactions as first-class objects that can be identified. Sections 6.2 and 6.3 describe the alternatives to agent coordination, and highlight the reasons supporting the choice of LCC as the formalism for specifying interactions. In particular, Section 6.2 describes coordination approaches centred on autonomous, rational agents. The first approach described involves modelling the mental states of the other agents and considering the exchanged speech acts as actions that changes these states. The other approach uses norms to specify the allowed, expected and forbidden behaviours of the agents. Section 6.3 describes service composition approaches, where the services are passive computational elements pulled together into workflows by some entities.

While the predictor depends on the coordination model used by the agents for their interactions, the predictions can be theoretically fed to any ontology matcher. Section

6.4 reviews the literature in ontology matching, presenting first the different categorisations of the mismatches between ontologies and of the matchers, then describing the basic matching techniques used by the available matching systems, and finally overviewing some of the most interesting projects.

Finally, Section 6.5 presents some of the ideas and techniques used in Natural Language Processing that inspired the working of the predictor.

6.2 Agent coordination and communication

In Chapter 2, we described inter-agent interactions via LCC, which constrain the agents to follow a predefined, stringent script. The literature also presents different approaches that give the agent varying levels of freedom and require different computational workloads. The *mentalistic* approach relies on agents modelling the internal state of the other agents, and planning interactions as sequences of actions, the exchanged messages, that change these internal states. The *social* approach is more oriented towards giving normative rules on what agents should do, without taking into account their internal state.

Applying the predictor presented in this thesis to the systems based on the mentalistic approach is difficult, because, in contrast with the use of choreographies, there is no defined context for an interaction. An interaction is the result of the involved agents planning their part of the dialogue. An agent, in order to recognise that it is in the same context, needs to match the current exchange of messages with previous exchanges: if the agent has participated in many different types of interactions, with some starting with the same subsequence, it cannot be sure which dialogue it is in until enough messages have been exchanged. Moreover, as different agents plan their part of the dialogue, each interaction can be different. With choreographies, on the other hand, agents agree to interact according to the shared interaction model, and this provide the stable context from the start of the interaction. The mentalistic approach also suppose a rational agent, able to reason over the received messages and decide the next steps, while our model does not make assumptions about the reasoning capabilities of the agents.

6.2.1 Mentalistic approach

In the mentalistic approach, speech actions are like actions: they change the state of the world, similar to physical actions [54]. Initial attempts such as [10] used formalisms like STRIPS: a speech acts could be defined by its preconditions and postconditions, expressed in multimodal logic, that were used to create plans. These early attempts were then refined into a more general theory by Cohen and Levesque [11]: speech acts are actions performed by rational agents that are trying to fulfill their intentions, according to their desires and current beliefs. The model is also called the Belief-Desire-Intention (BDI) model.

The speech act theory has influenced the development of various agent communication languages (ACL): we will overview KQML and the standardisation effort attempted by FIPA.

KQML

The *Knowledge Query and Manipulation Language* was initially developed in the early 90s as part of DARPA knowledge Sharing Effort to enhance the knowledge sharing and not specifically for agents.

KQML ACL aimed at creating a set of performatives to capture various propositional attitudes an agent wants to express. It has been developed to be independent of low level transport layer, as well as of the content language and ontology used.

A KQML message is composed by the locution and the content parts. The core of KQML is the speech act that wraps the content. The semantic of a message is expressed in terms of preconditions, postconditions and completion conditions. Conditions are expressed for both speakers and hearer of the utterance. Figure 6.1, taken and adapted from [67], shows a simple dialogue between an agent A, asking for the value of the attribute price (defined in an ontology called “travels”) of the flight BA786, and an agent B replying with the requested value.

FIPA ACL

The Foundation for Intelligent Physical Agent[1] is a standardisation body concerned with issues of interoperability. One of its committee is in charge of the development of ACL. FIPA ACL is similar to KQML: it is based on speech acts and it is BDI-centric. Also the syntax of the individual locutions resembles KQML.

[1] `http://www.fipa.org`

```
(evaluate
  :sender A :receiver B
  :language KIF :ontology travels
  :reply-with q1 :content (val(price BA786)))
(reply
  :sender B :receiver A
  :language KIF :ontology travels
  :in-reply-to q1 :content (= (price BA786) (scalar 225 pound)))
```

Figure 6.1: *Example of KQML dialogue*

```
(inform
   :sender agent1
   :receiver agent2
   :content (price BA786 225)
   :language sl
   :ontology travels
)
```

Figure 6.2: *Example of FIPA ACL message*

The specifications of messages provide an English description and a formal semantics, expressed in a form of Modal Logic called Semantic Language. The Semantic Language is a Multimodal logic able to represent certain and uncertain beliefs, desires and intentions.

Each communication act is is defined by its feasible preconditions and its rational effects. The feasible preconditions describe the appropriate mental state that the agent must have before sending the message, if it wants to comply with the standard. The rational effects specifies the expected mental state, given that the agent has performed the communication. The rational effects are usually defined for the recipient, but they do not need to hold in order to be compliant.

Figure 6.2, shows a simple message, sent from `agent1` to `agent2` to inform about the price of the flight `BA786`. Figure 6.3, shows the semantics for the messages `inform` and `request`. Both figures are taken from [67].

$\langle i, inform(j, \varphi) \rangle$

feasibility precondition: $B_i\varphi \wedge \neg B_i \left(Bif_j\varphi \vee Uif_j\varphi \right)$

rational effect: $B_j\varphi$

where $B_i\varphi$ means 'agent i believes φ', $Bif_j\varphi$ means that 'agent j has a definite opinion one way or another about the truth of falsity of φ', and $Uif_j\varphi$ means 'agent j is uncertain about φ; An agent i sending an `inform` *message with content φ respects the FIPA semantics if it believes φ, and it is not the case that it believes either that j believes whether φ is false or true, or that j is uncertain of the truth or falsity of φ.*

$\langle i, request(j, \alpha) \rangle$

feasibility precondition: $B_i Agent(\alpha, j) \wedge \neg B_i I_j Done(\alpha)$

rational effect: $Done(\alpha)$

where $Agent(\alpha, j)$ means that 'the agent of action α is j', and $Done(\alpha)$ means that 'the action α has been done. The agent i requesting agent j to perform action αmeans that agent i believes that the agent able to perform $\alpha$is j and that agent j does not currently intend that α is done.

Figure 6.3: *FIPA semantics of* `inform` *and* `request`

6.2.2 The Normative approach

Electronic Institutions

With Electronic Institutions the authors have tried to reproduce the way humans have developed social institutions, ranging from the state to private companies, to structure their social interactions within social institutions.

In eInstitution the interactions between agents are divided into scenes. In each scene an agent can take only one role. The scene is described as a Finite State Machine. The messages between agents causes the state of the interaction to change state. The interactions between agents are constrained by normative rules, that prescribe obligations and prohibitions for the agents in a particular situation. The scenes are connected together to compose a workflow, and the specification of the workflow describes how agents can legally move from one scene to another.

In eInstitutions agents and roles can be institutionals or externals. The institutional roles, and the agents that embody them, work to guarantee that the institutional rules are respected, while the external roles and agents are requested to conform to the institutional rules.

The institution prescribes a common language and a common ontology to use, but it makes no assumption about the internal structure of the agents.

> An Electronic Institution can be regarded as social middleware that sits between the external, participating agents and the chosen communication layer validating or rejecting their actions. [57]

There exists a tool, developed inside the OpenKnowledge project, for converting e-institutions into LCC.

6.3 Web Service composition

The mentalistic approach to agent coordination introduced before rely on autonomous, smart agents able to take decisions and to plan interactions involving other similar agents. The normative approach poses a lighter workload on the agents, as it reduces the search space for the actions forcing some behaviours and banning others. In a framework like OpenKnowledge, the norms are specified by interaction models, and the autonomy of the agents is reduced to the possibility of chosing what interaction to run.

However, in many applications the simpler integration and composition of distributed *services* may be enough, leaving the services unaware of their involvement in interactions.

While a service can be anything, the term is often used to indicate a *web service*, that is, according to W3C:

> "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."[7]

The services' preconditions and effects may be described with a rich ontology such as OWL-S, and a centralised planner composes them, either automatically or assisted by a human, creating a plan of execution. Alternatively, and more commonly, the plan may be designed a priori, as a centralised or distributed workflow of activities and the services are grounded, normally at design time, into those activities. We first introduce OWL-S in Subsection 6.3.1, and then overview two centralised and one distributed workflow languages in Subsection 6.3.2.

6.3.1 Semantic approach

OWL-S[2] is an ontology built on top of Web Ontology Language (OWL) by the DARPA DAML program as a replacement of the former DAML-S ontology. It is an ontology, written in OWL, for describing Semantic Web Services, with the aim of enabling users and software agents to automatically discover, invoke, compose, and monitor Web resources offering services, under specified constraints:

- *Automatic Web service discovery*: OWL-S aims at helping software agents to discover the Web Services that fulfill a specific need within some quality constraints, without the need for human intervention.
- *Automatic Web service invocation*: generally, it is necessary to write a specific program to invoke a Web Service, using its WSDL description. Using OWL-S a software agent should be able to automatically read the description of the Web Service's inputs and outputs and invoke the service.
- *Automatic Web service composition and interoperation*: in a Web where many services are available, it should be possible to perform a complex task, involving the coordinated invocation of various Web Services, based solely on the high-level description of the objective. OWL-S aims at helping in the composition and interoperation of the Services in order to enable the automatic execution of this task.

The OWL-S ontology is composed by the parts:

- *the service profile* describes what the service does. This information is primary meant for human reading, and includes the service name and description, limitations on applicability and quality of service, publisher and contact information.
- *the process model* describes how a client can interact with the service. This description includes the sets of inputs, outputs, pre-conditions and results of the service execution.
- *the service grounding* specifies the details that a client needs to interact with the service, as communication interaction models, message formats, port numbers, etc.

[2] `http://www.daml.org/services/owl-s/`

6.3.2 Web Service Workflow languages

A workflow is a:

> "reliably repeatable pattern of activity enabled by a systematic organization of resources, defined roles and mass, energy and information flows, into a work process that can be documented and learnt. "

Web services composition follows two alternative approaches: *orchestration* or *choreography*. Their primary difference is their scope. An orchestration model provides a scope specifically focussing on the view of one participant. Instead, a choreography model covers all parties and their associated interactions giving a global view of the system. The orchestration and the choreography distinctions are based on analogies: orchestration describes central control of behaviour as a conductor in an orchestra, while choreography is about distributed control of behaviour where individual participants perform processing based on outside events, as in a choreographed dance where dancers react to behaviours of their peers:

> "Dancers dance following a global scenario without a single point of control"[9]

In orchestration, a central process takes control and coordinates the execution of different operations on the involved web services. The web services do not know that they are involved in a composition process: only the central process is aware.

Choreography does not rely on a centralised coordinator: each web service knows when to execute its operation and with whom to interact. It is a collaborative effort focussing on the exchange of messages. All participants need to be aware.

In the following subsections first I overview two orchestration languages, one business-oriented (BPEL), and one more academic (YAWL), and then a choreography language (WS-CDL).

Choreographies are the approach used in this thesis. As we have seen in Section 2.4.4, LCC was used to specify the interactions. While another choreography language such as WS-CDL could have been used, LCC is more compact and directly executable. These advantages lead to its choice.

Orchestrations define the behaviour of a single agent: the predictor can be used only for that agent. The other agents are not aware of being part of the interaction: their services are invoked from the orchestrating agent without any reference to an interaction context. If they wanted to use the predictor, they would need to recognise,

from the sequence of invocations, to be in a specific type of interaction. The problem is similar to the one encountered by agents using a mentalistic approach: each invocation can increase the probability of being in a certain type of interaction, but it may requires several messages to reach a certain level of confidence. Moreover, different orchestration agents may use different workflows for the same goal, possibly changing the invocations that an agent providing a set of services can expect.

BPEL (Business Process Execution Language)

BPEL (Business Process Execution Language) for Web services is an orchestration language. It is an XML-based language designed to enable task-sharing for a distributed computing or grid computing environment - even across multiple organisations - using a combination of Web services. Written by developers from BEA Systems, IBM, and Microsoft, BPEL combines and replaces IBM's Web Services Flow Language (WSFL) and Microsoft's XLANG specification.

A BPEL process receives a request and to fulfill it it invokes the involved web services and then responds to the caller. Defining a BPEL process is essentially defining a new web service that is the composition of existing services. A BPEL process consists of steps: each step is called *activity*, that can be primitive or structure. A primitive activity can be an invocation of a web service, waiting a reply from an asynchronous call, generating responses for synchronous operations, manipulating variables, indicating faults, waiting specified intervals, terminating the process. A structure activity is a composition of primitive ones. Primitive activities can be composed in sequence, in parallel, in loops, or as branches with conditions.

YAWL (Yet Another Workflow Language)

In recent years many different workflow products have appeared, each with its own semantics and constructs. The task of comparing them has induced researchers, in particular those in Van der Aalst group in the Eindhoven University, to identify the most frequently used patterns applied in the development of workflows [64]. The workflow patterns are pragmatically used to compare the expressivity of the different workflow languages. A more formal foundation to represent and compare workflow is provided by Petri nets, even though some patterns are difficult to represent even with extended Petri nets. To overcome these difficulties, the Van der Aalst groups has developed another workflow language, YAWL [62], based on patterns, and defined in

terms of a transition system. A workflow specification in YAWL is a set of process definitions which form a hierarchy. Tasks are either atomic tasks or composite tasks. Each task refers to a process definition at a lower level in the hierarchy. Atomic tasks are leaves of the graph-like structure.

WS-CDL (WS-Choreography Description Language)

The Web Service-Choreography Description Language [34] is a specification by the W3C defining a XML-based business process modeling language that describes common and collaborative observable behaviour of multiple services that need to interact in order to achieve some goal. WS-CDL describes this behaviour from a global or neutral perspective rather than from the perspective of any one party. WS-CDL is a description and not an executable language.

Peer-to-peer protocols described in WS-CDL do not have a centralised point of control: each party remains autonomous and no party is master over any other. There are no global variables, conditions or workunits, as it would require centralised storage and orchestration. WS-CDL permits a shorthand notation to enable variables and conditions to exist in multiple places, but this is syntactic sugar to avoid repetitive definitions. There is also an ability for variables residing in one service to be aligned (synchronised) with the variables residing in another service, giving the illusion of global or shared state.

In WS-CDL all messages are described as information types and there is no distinction between application and infrastructure messages. All that WS-CDL describes is the ordering rules for the messages which dictate the order in which they should be observed. When these ordering rules are broken WS-CDL considers them to be out-of-sequence messages and this can be viewed as an error in conformance of the services that gave rise to them against the WS-CDL description.

Services are any form of computational process with which one may interact, examples are a buying process and a selling process that are implemented as computational services in a Service Oriented Architecture (SOA) or as a Web Services implementation of an SOA: WS-CDL is not explicitly bound to WSDL and therefore it can play the same global model role for both SOA services and Web Services. It is possible to use WS-CDL to describe a global model for services with no WSDL descriptions (they can have Java interfaces) as easily as it is to describe services that do have or will have WSDL descriptions.

6.4 Ontology Matching review

We have seen in Section 3.2 how the success of the ontologies has brought a wealth of ontologies, not their standardisation. We have presented how this heterogeneity is tackled using Ontology Matching algorithms. In this section we will first introduce different classifications for the sources of mismatches between ontologies in Subsection 6.4.1 and for the matching algorithms in Subsection 6.4.2, then overview the elementary matching techniques in Subsection 6.4.3 and finally review a group of interesting projects in Subsection 6.4.4.

6.4.1 Ontology mismatches classifications

Hameed, Preece and Sleeman [31] distinguish three perspective in the classification of mismatches.

Knowledge Representation Perspective

According to [65], ontologies can differ because of two main categories of mismatches: *conceptualisation* and *explication* mismatches. The first category of mismatches originates from the initial phase of conceptualisation of the domain. Conceptualisation mismatches include class and relation mismatches: for example, classes can be divided into different subclasses (for example, the class *animal* can be subclassed into *mammals*, *birds*, *reptiles*, *fishes* in one ontology and into *herbivores*, *carnivores* and *omnivores* in another), or attributes can be assigned to different classes (for example, two ontologies can have the same classes *camera* and *digital_camera*, the second subclass of the first, and the attribute *lens* may be attached to *camera* in one ontology and to *digital_camera* in the other). Explication mismatches are caused by differences in the way the conceptualisation is specified in a formal language: for example, there might be ambiguities derived from using the same term to identify different entities (for example, *bank* meaning *financial institution* in one ontology and *ridge* in another), or from using different terms to identify the same entity (for example, *car* and *automobile*).

Database perspective

Wiederhold [66] proposes a different set of mismatches, more oriented to data sources:

key difference: different naming for the same concept

scope difference: distinct domains, or distinct coverage of domain members

abstraction grain: varied granularity of detail among the definitions

temporal basis: mismatches concerning time, periods, intervals

domain semantics: distinct domains, and the way they are modelled.

value semantics: differences in the encoding of values (date format, currencies,...)

Knowledge Elicitation Perspective

Shaw and Gaines [55] described four dimensions to map knowledge elicitation situations likely to be encountered when experts are involved in the process of developing a knowledge-based system:

Conflict: when experts use the same term for different concepts

Correspondence: when the experts use different terms for the same concept

Constrast: when the experts use different terms and have different concepts

Consensus: when all the experts use the same term for the same concept

6.4.2 Matchers' Classifications

Different ontology mapping surveys have been compiled through the recent years [56, 49, 33]. They offer a classification of the ontology matching systems and a review of the techniques at the state of the art.

Shvaiko and Euzenat, in their [56], distinguish three dimensions for the classification:

input dimensions: these dimensions are about the kind of input on which an algorithm operate:

- the data/conceptual model in which the ontologies are expressed (E-R schemas, OO structures, XML, RDF or OWL ontologies)
- the type of data that the algorithm exploits for finding correspondences: schema data (the conceptual model of the ontology), instance data, or both

process dimensions: the type of computation involved, that can be either exact or approximate

output dimensions: what result is returned to the user: one-to-one correspondences between the entries in the ontologies, graded or all-or-nothing answers, and the kind of relations that between the entries (similarity, equivalence, subsumption, ...)

6.4.3 Elementary matching techniques

Most ontology matchers combine the results produced by elementary matchers. The elementary matchers can be classified in many different way. Shvaiko and Euzenat propose two classifications, based on:

granularity and input interpretation that divides the matchers in *element-level* ones, that analyse the entities in isolation ignoring their relations with other entities, or *structure-level* matchers, that analyse how entities appear together in a structure

kind of input that divides the matchers based on the type input (syntactic, external or semantic)

Element-level techniques

String-based techniques They consider names, labels and comments as sequence of characters. Often the strings are normalised before being compared: they are converted to lowercase, characters with diatric symbols (such as accents or cedillas) are replaced with their more common versions (*é* to *e*, *ń* to *n*, etc), spaces are trimmed, and finally hyphens, apostrophes, punctuation symbols or digits are removed.

- substring: verifies if one string is a substring of another (can be a prefix, as in in *integer* and *int*, a postfix, as in *telephone* and *phone*)
- Hamming distance: counts the number of positions in which the two strings differ. For example, *synchronise* and *synchronize* have an Hamming distance of 1.
- edit distance: takes two strings and counts the minimum number of insertions, deletions, substitutions of characters required to transform one string into another (usually normalised by the length of the longest string). For example, *article* and *aricle* have a distance of 0.14, while *article* and *paper* have a distance of 1.

- *n*-gram: takes two strings and counts the number of common *n*-grams (sequences of *n* characters). For example, *article* and *aricle* have a similarity of 0.5, *article* and *paper* a similarity of 0 while *article* and *particle* have a similarity of 0.83.

Comparing only the labels of the entities in two ontologies cannot handle synonyms (different words that name the same entity) and homonyms (same word used to name different entities).

Language-based techniques In order to deal with the problems caused by synonyms and homonyms, more sophisticated matchers consider words in label to have a structure and a meaning, derived by their use in some natural language. Euzenat and Shvaiko distinguish between *intrinsic* and *extrinsic* techniques.

In intrinsic techniques, the text is normalised to reduce the form to a standard form that is more easily recognised:

- tokenisation: is the process of demarcating and possibly classifying sections of a string of input characters. For example, the sentence "*advances in imaging technology*" becomes the list of strings <"*advances*", "*in*", "*imaging*", "*technology*">.

- lemmatisation: strings of tokens are morphologically analysed to reduce them to a normalised, standard form. In many languages, words appear in several inflected forms: for example, in English, the verb *'to walk'* may appear as *'walk'*, *'walked'*, *'walks'*, *'walking'*. The base form, *'walk'*, that one might look up in a dictionary, is called the lemma for the word. The list <"*advances*", "*in*", "*imaging*", "*technology*"> would become <"*advance*", "*in*", "*image*", "*technology*">.

- elimination: words that carry little meaning (like articles or prepositions) are dropped. For example, in the list above the token "*in*" would be dropped yielding <"*advance*", "*image*", "*technology*">.

- term extraction: morphologically similar phrases are recognised, using patterns learnt from large corpora. This is normally obtained identifying the role of the words (whether they are noun, verb, ...) and then comparing the resulting structures. For example, $\texttt{Noun}_1\,\texttt{Noun}_2$ and $\texttt{Noun}_2\,\texttt{of}\,\texttt{Noun}_1$ are considered equivalent, and therefore "*newspaper article*" would be considered equivalent to "*article of newspaper*".

In extrinsic techniques, use external common knowledge or domain specific thesauri to match the entities:

- lexicons: or dictionaries, are set of words with a definition in natural language.
- multi-language lexicons: are dictionaries where the definition is replaced by a word in another language
- thesauri: are lexicons where the relations between words are made explicit. One of the most commonly used thesaurus is WordNet [40].

Extrinsic techniques help in dealing with synonyms. However, words are often used with different meanings, and a resource such as a theaurus can show incorrect relations, increasing the false positives and consequently decreasing precision. To deal with this problem the words used in labels need to be disambiguated, restricting the senses to those consistent with the context. The probability distribution computed by the predictor can help here, providing additional contextual information.

Alignment reuse They store alignment used in previous matching, assuming that many ontologies or schemas can be similar to previously matched ones.

Structure-level techniques

Internal structure techniques Deal with internal constraints applied to definitions of the entities: data types, cardinality of attributes,...

Graph-based techniques They consider the input ontologies as labelled graphs, and are based on the intuition that if two nodes in two ontologies are similar, then their neighbours will be likely similar.

- graph matching: searches the maximally common directed subgraphs
- children matching: the similarity between two inner nodes is computed based on the similarity of their children nodes
- leaves: the similarity between two inner nodes is computed based on the similarity of their leaves nodes
- relations: the similarity between two nodes is computed based on their relations with other nodes (properties)

Semantic based techniques

In a semantic method the model-theoretic semantics is used to justify the results [16]: deductive methods are used on preprocessed ontologies.

Upper level Ontology The lack of common ground between the ontologies to map is covered by upper level formal ontologies like SUMO [41] or DOLCE [42] to provide a logical based system that the matcher can use to reason about the correspondences.

Deductive techinques They give a semantic interpretation to the ontologies, and use well grounded deductive methods:

- SAT based: decompose the tree to a set of node matching problems, translating each node matching into a propositional formula
  $axioms \rightarrow rel(context_1, context_2)$
  and check the validity of the formula. The axioms encode the background knowledge
- Description-Based techniques: overcome some of the limitation of the SAT based approach

6.4.4 Projects review

Following the classification used in [16], we divide the overview of the projects into those *schema based* and those *instance based*. A project is schema based when it exploits mainly the conceptual definitions of the ontologies to find the correspondences, while it is instance based when it uses the instances of the ontology for the comparisons.

Schema based

Mafra Developed by Maedche, Silva and Rocha [44], MAFRA is oriented to help human users to map ontologies from different institutions.

The conceptual framework divides the process of matching two ontologies into five steps, and four transversal tasks. The process starts by trying to render uniform the language in the source and the target ontologies. Once the syntactic and lexical heterogeneity have been reduced, it proceeds to discover the similarity between the entities in the ontologies using a multi-strategy and multi-algorithm process that analyses

both the lexical and the property similarity of terms. Once the similarities have been computed, they are used to create semantic bridges between the entities in the source and target ontologies. Then the process continues, evaluating the semantic bridges and transforming the instances from the source ontology to the target ontology. Finally, post-processing is executed to improve the alignment.

Similarity Flooding It uses an hybrid matching algorithm based on similarity propagation. Consider the schemas as directed labeled graphs. The technique starts from a string-based comparison between nodes in order to find an initial alignment. It then iterates, spreading the similarity from similar nodes to adjacent neighbours through propagation coefficients. The similarity increases until the fix point is reached.

It consider the alignment as a solution to a clearly stated optimisation problem.

S-Match The S-Match project [23] has been developed by Giunchiglia and Shvaiko at the University of Trento. It takes two trees, and computes the strongest semantic relation between each pair of nodes.

The process is organised into four macrosteps:

1. Compute concepts of labels, for all labels in the two trees. A concept of a label is obtained by first tokenising labels, then lemmatising the resulting tokens and finally using an oracle (WordNet in this case) to obtain the senses of lemmatised tokens. Different senses are combined in a disjunction to form a propositional formula for each label. Tokens from expression like “wines and cheeses” form a disjunction ($wine \vee cheese$), while terms from expression like “Italian cheeses” form a conjunction ($italian \wedge cheese$)

2. Compute concepts at nodes as the conjunction of the concept of label formulae in the concept path to root

3. Compute semantic relations between pairs of labels from the two trees

4. Compute semantic relations between pairs of nodes from the two trees

The semantic relations between pairs of labels are used as input for computing the relations between nodes. The system tries to verify the formula:

$$axioms \rightarrow rel\left(context_A, context_B\right)$$

The *axioms* are the computed relations between labels, while $context_A$ and $context_B$ are the concepts at the nodes. As the propositional solvers are satisfiability checkers, the formula is then converted to:

$$axioms \wedge \neg rel\,(context_A, context_B)$$

In [24] the developer of S-Match present an improved version of their work, that exploits the structure of the formulae above to increase the speed of satisfiability computation. The optimised version of S-Match is particularly efficient on large classification, where it perform much better than COMA and than the original version of S-Match, and it requires much less memory than Similarity Flooding.

COMA/COMA++ The COMA project [13] is a schema matching system, and can be applied to XML or databases schemas. The schemas are translated to directed acyclic graphs that are then compared to find correspondences. The central idea in COMA is to combine different matching algorithms to find better results. Matching is an interactive and iterative process, composed by three main steps:

- **Optional user feedback:** the user can manually provide match correspondences, confirm or reject proposed matches
- **Execution of matchers:** multiple matchers are used independently to obtain several similarity measures. Matchers can be simple, hybrid or reuse-oriented.
- **Combination of individual match results:** the results are aggregated into a combined value for each pair, using some strategy (like the average or the maximum of the results), and then the candidates with the best similarity values above a threshold are chosen.

COMA introduces also the reuse of past alignments, in the form of whole schemas or fragments of them.

COMA++ [1] extends COMA improving the graphical interface for a better user interactivity, improving the reuse of past alignments and replacing the internal representation language to support schemas and ontologies written in different languages.

Instance based

Glue Glue [14] combines different machine learning techniques to find correspondences. The matching is based on a representation of similarity between concepts

formally defined as their joint distribution. The similarity between two concepts A and B is given by their joint distribution $A \cap B$. Computing the joint distribution means finding instances that belongs to both concepts A and B. Usually instances of the two concepts are separated: to solve this problem they use machine learning to develop two classifiers for the instances.

The instances of A are used to create a classifier for A, that is then used to classify the instances of B, and vice versa. Deciding which learning algorithm to use and which information to exploit is difficult, and therefore a multi learning strategy is used. The predictions supplied by the algorithms are then combined by a meta-learner.

Available domain constraints and general heuristics is also used to improve accuracy.

Mixed approach

QOM The QOM project [15] addresses the problem of efficiency in ontology mapping and considers the trade off between efficiency and quality. This is done introducing the idea of filtering correspondence candidates that are unlikely to be verified.

The matching process is iterative, and the main steps are:

- **Selection of candidate pairs** whose similarity should be checked: candidates are selected using different strategies to classify them into more promising and less promising ones. The strategies can use the labels of the pairs (only similar ones are kept), the hierarchy of the ontology (the ontologies are mapped from the top down), the result of previous iterations (only terms close to terms mapped in the previous iteration are mapped) or a combination of these.

- **Similarity computation:** the similarity is computed using a range of similarity functions. These can measure the string similarity of the labels, can check if the concepts share the same properties, the same descendants, the same siblings...

- **Similarity aggregation:** the measures given by these functions are then combined. The candidates with low aggregate measures are discarded, then bijective candidates (candidates for which the relation can work in both directions) are kept and finally the candidates with the strongest aggregate measure are kept.

These steps are repeated until no new correspondence can be found.

6.4.5 Approximate Structure-Preserving Semantic Matching

Most of the projects described above aim at finding correspondences between terms in ontologies. In open systems, such as OpenKnowledge, they can be used to map the *content* of messages, or the content of invocations to web services. However, it is often necessary to adapt structures: for example, in OpenKnowledge, peers need to map their methods to the constraints in the interaction models: parameters can be called with different names, might be in different positions, or their structure might be different. Similarly, it might be necessary to dynamically map the invocation of a web service, as defined in a workflow, to the WSDL interface of the web service.

Often it is not possible to map exactly every element in the two structure: however, it can be enough to be able to invoke the service, possibly with some parameters set to a default value. The work presented in [21] deals with the problem of approximate matching of structures. Web services are considered first order predicates, and are transformed into trees. Two trees are matched, extracting the correspondences between the nodes and evaluating whether they are similar enough.

The matching is performed in two steps: first the nodes are matched, and then the trees. Node matching considers only the labels at the nodes, and the context provided by the tree. It uses S-Match, described above, to find the relations between the nodes of the two trees: the concept at each node is expressed as a logical formula, and the relation is verified using a SAT algorithm.

The correspondences found by node matching are then filtered using abstraction theory. Abstraction theory categorises the type of abstraction operations. Among the them, some operations provide the only ways to alter two first-order terms changing their signature but maintaining completeness. Some of the correspondences found in the first step do not represent these operation: therefore it may happen that functions are wrongly mapped to variables, or variables to functions. These correspondences are dropped, leaving only those that maintain completeness.

Tree edit distance is used on the allowed correspondences to compute the similarity between the trees. In its formulation, tree edit distance consider the basic operations that can be applied to a tree to change it into another tree: addition, removal and replacement of a node. The abstraction operations seen above are mapped to these basic operations, and a cost is assigned to them. The algorithm computes the minimal cost of transforming one tree into another.

At the end of the whole procedure, we have a set of correspondences between

nodes (which can be interpreted as correspondences between parameters), and a value that summarises the similarity between the trees. If the similarity is above a certain threshold, the matching is considered valid, and the correspondences can be used.

In OpenKnowledge, as we have seen in Section 2.7, this procedure is used to evaluate the capability of a peer to perform an interaction model by comparing the constraints in it with the peer's services, and to create the adaptors used during the run of the interaction to call the services provided by the peer when constraints are met.

6.4.6 Dynamic Ontology Refinement

This approach, developed by Fiona McNeill, Alan Bundy and Marco Schorlemmer at the University of Edinburgh [39], tries to tackle the failures in plan execution due to mismatch of ontologies between the involved agents. The aim is to improve the robustness of planning, adapting the theory behind the decisions after failures. It is not exactly an ontology mapping system, but it deals with interaction among agents that do not share the same ontology.

In this model the plan is accompanied by a justification of every step. The justification is produced by a "plan deconstructor" that analyses the plan produced by the planner and explains the theory that motivate each step. The theory is the knowledge of the world that the agent has, represented by its ontology.

If the execution of the plan fails, the agent tries to find the exact point in the plan where the failure has occurred, and then tries to understand how the justification for the step caused the failure. For example the ontology might have oversimplified the domain, and thus it might have justified a wrong decision.

Then, if possible, the agent tries to refine the ontology, possibly interacting with the other agent, to adapt it better to the domain, and repeat the communication process. In the current version, the changes yielded by the refinements are permanent.

6.5 Natural Language Processing

Some of the ideas at the basis of the work presented in this thesis were inspired from the field of Natural Language processing. Dialogue norms and conventions appear at syntactic level: a request is normally followed by an answer, an offer by an acceptance or a rejection. The intuitions about syntactic norms has prompted researchers in NLP to study the possibility of *dialogue grammars*, which have often been represented as

finite state machines, where the speech acts are the transition states between admissible states of the dialogue.

Another source of inspiration has been the use of Markov models to predict information about portions of text given the information collected up to the portion. The information can be the part-of-speech of a word as described in the next Subsection, or the type of speech act in dialogues, as discussed in Subsection 6.5.2. Even though the predictor presented in this thesis does not use a Markov model, as discussed in Chapter 4, it represents a useful comparison.

6.5.1 Part-of-speech tagging

One of the tasks required for parsing and understanding natural language is to tag each word in a sentence with its appropriate part of speech, that is whether a word is a verb, a noun, an adjective and so on. One of the techniques used for tagging [37] is based on Markov model. The sequence of tags in a text is considered as a Markov chain, and assumes that a word's tag only depends on the current word and on the previous tag. It also assumes that the dependency does not change over time.

6.5.2 Dialogue translation

For example, in automatic dialogue translation in face-to-face situations, the ability to predict the dialogue speech acts can improve the results: in [50], a corpus of manually tagged dialogues is analysed in order to extract the posterior probability of a speech act d_j given the history of the previous acts $d_{1\ldots j-1}$. Since it is impossible to determine the probability of arbitrarily long sequences, they use *n-grams*: only N previous speech acts are used: $d_{j-N+1}\ldots d_{j-1}$. In the paper they analyse the possibility of using a dialogue grammar, in the form of a Finite State Machine that encodes the state of a dialogue (starting phase, end, proposal or reaction). First, they tried to exploit the knowledge provided by the grammar by training directly the grammar attributing probabilities to the states and to the transitions, but this approach yielded results consistently worse than the simple statistical one. Then, they included the knowledge in the interpolation formula, and then they replaced old dialogue acts with states: since the number of states is less than the number of speech acts, they were able to cluster more results with the same dataset.

Finally, they exploited the knowledge of the speaker: they tagged each speech act with the contributing speaker, making explicit the direction of the acts: if speaker A

poses a question, and then A makes a further utterance, it is likely to be an explanation or a correction; if the second utterance is produced by speaker B, then it is likely to be a reply.

6.6 Summary

In Chapter 2 we introduced the concepts relevant to the work presented in the thesis, in particular those related to the communication between agents and to the problem of tackling heterogeneity in the communication. In this Chapter we overviewed the various approaches available in literature.

We have first described the mentalistic and the social approaches to communication between autonomous agents; we have then moved towards the composition of passive services, either by planning using rich services' descriptions, or by designing a workflow of activities grounded to the services. The OpenKnowledge project described in Section 2.7 lays between the two models: the peers are proactive in the choice of pre-defined interactions.

We then analysed the problem of Ontology Matching. First, we presented the classifications available in literature for the source of mismatches between ontologies, and the classifications used in the main reviews for the ontology matching algorithms. Second, we described the basic techniques used in the matching algorithms, and finally we overviewed a set of interesting and relevant projects.

Chapter 7

Conclusion

We increasingly require software applications to interact one with another: they are becoming access points for services distributed in the network, working as providers or brokers for these services. However, applications are written by different developers with different goals in mind, and they also evolve over time: their main common feature is their diversity.

The idea behind the semantic web is to define these services and the data they process using a machine-readable language, defined in an ontology, in order to find and combine them automatically. However, while there has been a slow but steady adoption of a small set of common syntaxes (such as RDF or OWL), there has been no agreement over the semantics used: many different ontologies, most of them written in RDF or OWL, are used to describe the services and their data.

To overcome this heterogeneity, a variety of ontology matching algorithms have been developed. They aim at statically matching two or more ontologies, finding all the possible correspondences between them. However, when the aim of the matching is to allow communication between agents, they do not exploit the additional information provided by the context of the interaction itself. This additional information can improve efficiency, by removing the need to compare terms likely to be unrelated to the interaction, and can improve both completeness (recall), often low because of a lack of domain-specific information, and correctness (precision), by reducing ambiguities that a lack of context normally bring.

The work presented in this thesis is a system that first analyses the history of similar interactions in order to create a statistical model of one type of interaction and then uses this model to compute a probability distribution for the content of the exchanged messages in new interaction runs. The probability distributions can be forwarded to

an ontology matching algorithm that focuses its computational effort on verifying the suggested hypotheses, without wasting time on evaluating correspondences not related to the interaction. The model is updated feeding back to the predictor the results of the matching process.

The model is based on two main assumptions about the content of the messages: the terms in the messages appear with a frequency reflecting a probability distribution in the community of users and the context of the interaction model itself; the terms in messages may have relations with other terms in previous messages. The relations can be simple correlations, can be implicit or explicit ontological relations that the system is able to understand.

In the introduction (Section 1.1), I stated that this thesis had two key goals:

1. improve the efficiency of an arbitrary ontology matcher,
2. maintain or improve the quality of the matcher's results

Both goals have been reached: the evaluation of the proposed method shows that a relatively small number of interaction is often enough to obtain a remarkable improvement of the efficiency of the matcher (about 10 times quicker), while keeping precision and recall close to the same values of the baseline model that does not use the predictor. A problem discovered during the evaluation process is that a wrong probability distribution can sway the matcher, decreasing both precision and recall. This happens during the initial period, when the model is still unstable and imprecise: after this period, the computed distribution tends to reflect the actual distribution. The tests have shown that, if even if we trade off precision for efficiency, recall remains higher than the baseline.

The main requirement is to use a framework that allows the description of the interaction sequence: workflow based systems provide the functionality, but are often centralised. With the OpenKnowledge project we have shown that these results can be obtained in a purely peer-to-peer environment.

7.1 Future work

During the work presented in this thesis I had to decide which areas to cover more in detail, and which areas to leave out for lack of time and space. During the development, limitations were identified and I often had to opt for simplifications, as the solutions, although intellectually interesting, had implications too vast to be tackled in a single

thesis. This section tries to present some of the ideas for future work that could extend and improve the current state of the system.

Drop assertions that are not consistent.

When the predictor needs to instantiate the statistical model to the current interaction in order to compute the probability distribution of terms for a variable, it only drops assertions whose condition ζ is not consistent with the current state of the interaction. For example, as we have seen in Section 4.4, assertions about the posterior probability of the offer being about a compact car, given that we asked for an accommodation, are removed. However, assertions about the prior probability of the offer being a compact car are not removed. Introducing a basic reasoner that removes, or discounts, assertions about terms considered to be inconsistent might improve the performance of the predictor.

Matching different interactions

One of the limitation of the work presented here is that the model of an interaction is strictly bound to one interaction model. Over time, the peer will create many of these models for the different interactions it is involved in. However, if the interaction model used for a particular task changes, the knowledge collected on the previous version of the interaction becomes useless: the peer has to start creating a new model.

Therefore, recognising similar interactions would be an interesting development. When a peer starts an interaction it has never seen before, it could match it against all those previously encountered, possibly finding one or more similar. Then, it could use the information contained in the corresponding models, weighted by some measure of confidence in the similarity, to predict the content of the new interaction.

Extending ontologies

It was suggested as one of the applications of the predictor in Section 3.5, but it was not analysed in detailed. The assertions about ontological relations can be used to drive the extension of the ontology when failures to find mappings occur. When a message arrives with a foreign term w_j that does not correspond to any known term t_i in the peer's ontology, it can be possible to verify what were the ontological relations in the model that the term should have most likely satisfied. If the same event takes

$$
\begin{array}{l}
\texttt{a}(\texttt{customer},\texttt{C}) :: \\
\left(\begin{array}{l}
\texttt{null} \leftarrow \texttt{wantSLR}() \\
\texttt{then} \\
\texttt{slr\_model}(\texttt{Brand},\texttt{Model}) \Rightarrow \texttt{a}(\texttt{camera\_vendor},\texttt{V}) \\
\quad \leftarrow \texttt{brand}(\texttt{Brand}) \text{ and } \texttt{model}(\texttt{Model})
\end{array}\right) \\
\text{or} \\
\left(\begin{array}{l}
\texttt{compact}(\texttt{Brand},\texttt{Model},\texttt{Lens} \Rightarrow \texttt{a}(\texttt{camera\_vendor},\texttt{V}) \\
\quad \leftarrow \texttt{brand}(\texttt{Brand}) \text{ and } \texttt{model}(\texttt{Model}) \text{ and } \texttt{lens}(\texttt{Lens})
\end{array}\right)
\end{array}
$$

Figure 7.1: *Specific interaction model*

place with a certain frequency, the system could suggest that a new term, satisfying the ontological relations in the model, should be added to the ontology.

Expressivity of ontological relations

At the moment only basic ontological relations are used by the ontological strategies: `subclassOf`, `superclassOf`, `siblingOf`, `propertyOf`, `domainOf` and `rangeOf`. An interesting development could be to increase the expressivity of the relations in the assertions. However, the search strategy should be revised: at the moment, all the possible alternative relations are verified, but it would be unfeasible if the set of relations grows due to the increased expressivity. Some heuristics in the choice of the alternative relations to evaluate should be found.

Types of dialogues and predictor usefulness

The predictor helps *run-time* (also called *on-line*) matching: it helps matching terms that arrive in messages during the execution of an interaction. Not all interactions benefit from using the predictor: interactions where the content of messages is strictly defined before the run do not gain from the predictor. In these interactions most of the matching is off-line (for example, between constraints and the methods in the plug in components used in OpenKnowledge).

A specific interaction about buying a digital camera like the one shown in Figure 7.1 is strictly defined. This interaction has constraints for obtaining resolution, type of lens, brand, and so on. In a model like OpenKnowledge, it means to match at subscription time these constraints with the methods in the plug-in components locally

$$
\begin{array}{l}
\texttt{a}(\texttt{customer},\texttt{C}) :: \\
\texttt{ask}(\texttt{Product}) \Rightarrow \texttt{a}(\texttt{vendor},\texttt{V}) \leftarrow \texttt{want}(\texttt{Product}) \\
\texttt{then} \\
\texttt{a}(\texttt{c\_refine},\texttt{C}) \\
\\
\texttt{a}(\texttt{c\_refine},\texttt{C}) :: \\
\left(\begin{array}{l}
\texttt{inquire}(\texttt{PProperty}) \Leftarrow \texttt{a}(\texttt{v\_refine},\texttt{V}) \\
\texttt{then} \\
\texttt{definition}(\texttt{PProperty},\texttt{Value}) \Rightarrow \texttt{a}(\texttt{v\_refine},\texttt{V}) \\
\quad \leftarrow \texttt{define}(\texttt{PProperty},\texttt{Value}) \\
\texttt{then} \\
\texttt{a}(\texttt{c\_refine},\texttt{C})
\end{array} \right) \\
\texttt{or} \\
\texttt{offer}(\texttt{Offer}) \Leftarrow \texttt{a}(\texttt{v\_refine},\texttt{V}) \\
\vdots
\end{array}
$$

Figure 7.2: *Generic interaction model*

installed in a peer, as described in Section 2.7. These constraints will be satisfied by providing very specific information, possibly only numerical values (resolution) or elements from a list known a priori (available brands for a camera). Matching is mainly offline, performed both for finding the proper interaction to run (I need to buy a digital camera, not an analogue camera), and then to bridge the constraints with the peers' capabilities (methods in the OpenKnowledge, as we have just seen).

A more generic interaction about buying a product, like the one shown in Figure 7.2, requires more run-time matching: constraints have to be more generic, and some of the requests are defined at run-time. For example, what attributes should be asked to a customer depends on what is asked by the customer, and cannot be defined a priori. The offline matching is rather minimal, while most of the work has to be performed at run-time.

In the first case, the interaction model designer enforces a strict semantics, in the second case the community of users will define the semantics by using it.

It would be interesting to study how different interaction specifications can influence the usefulness and the efficacy of the predictor.

Appendix A - Formalisms and Conventions

Font use

- LCC code, LCC variable names and LCC constraints are written in typewriter font: `Product`, `refine(Product,Refinement)`, ...
- The content of LCC variables, usually terms from one of the peers' ontologies, are written in italics and surrounded by quotes: *"accommodation"*, *"hotel"*,...

Ontology mapping

- An ontology is represented by O_i where the index i refers to the origin of the ontology:
  - In the examples, O_a is the agent's ontology, while O_r is the ontology of interaction run, formed by the union of the terms used by the different agents.
- w_i is the term to be mapped from a foreign ontology to a term t_j in the local ontology

Probability

- $P(x_i)$ is the probability of the event x_i
- $\mathbf{P}(\mathbf{X})$ is the probability distribution of a random variable $\mathbf{X}$, and corresponds to the vector:
  $\mathbf{P}(\mathbf{X}) = \langle P(X = x_1), \ldots, P(X = x_n) \rangle$

Sets

- A set is written with a Greek or Latin capitalised letter: Ψ, M.
- The symbol $|\Psi|$ is used to indicate the cardinality of Ψ: if $\Psi = \{a, b, c\}$, then $|\Psi|$ is 3

Bibliography

[1] D. Aumueller, Hong-Hai Do, S. Massmann, and E. Rahm. Schema and ontology matching with coma++. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 906–908, New York, NY, USA, 2005. ACM Press.

[2] F. Baader, D.L. McGuiness, D Nardi, and Patel-Schneider P., editors. *Description Logic Handbook: Theory, implementation and applications*. Cambridge University Press, 2002.

[3] A. Barker and B. Mann. Agent-based scientific workflow composition. In *Astronomical Data Analysis Software and Systems XV*, volume 351, pages 485–488, 2006.

[4] P. Besana. A framework for combining ontology and schema matchers with dempster-shafer. In *Proceedings of the 1st International Workshop on Ontology Matching (OM-2006)*, volume 225. CEUR-WS.org, 2006.

[5] P. Besana and D. Robertson. Probabilistic dialogue models for dynamic ontology mapping. In *Proceedings of the Second ISWC Workshop on Uncertainty Reasoning for the Semantic Web*, volume 2. CEUR-WS.org, 2006.

[6] P. Besana and D. Robertson. How service choreography statistics reduce the ontology mapping problem. In *ISWC2007*, 2007.

[7] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web services architecture. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/, February 2004. W3C web site.

[8] P. Bourque, R. Dupuis, A. Abran, J. W. Moore, and L. Tripp. The guide to the software engineering body of knowledge. *IEEE Software*, 16(6):35–44, 11-12 1999.

[9] M. Carbone, K. Honda, and N. Yoshida. Structured global programming for communication behaviour. http://www.pi4tech.com/xwiki/bin/view/research/papers, 2006.

[10] P. R. Cohen and C. R. Perrault. Elements of a plan based theory of speech acts. *Cognitive Science*, 3:177–212, 1979.

[11] P.R. Cohen and H.J. Levesque. Rational interaction as the basis for communication. *Intentions in Communication*, pages 221–256, 1990.

[12] Li Ding, T. Finin, A. Joshi, R. Pan, R. Scott Cost, Y. Peng, P. Reddivari, V.C. Doshi, and J. Sachs. Proceedings of the thirteenth acm conference on information and knowledge management. In *Proceedings of the Thirteenth ACM Conference on Information and Knowledge Management*, 2004.

[13] Hong Hai Do and E. Rahm. Coma - a system for flexible combination of schema matching approaches. In *VLDB*, pages 610–621, 2002.

[14] A. Doan, J. Madhavan, R. Dhamankarse, P. Domingos, and A .Halevy. Learning to match ontologies on the semantic web. *The VLDB Journal*, 12(4):303–319, 2003.

[15] M. Ehrig and S. Staab. Qom - quick ontology mapping. In *International Semantic Web Conference*, pages 683–697, 2004.

[16] J. Euzenat and P. Shvaiko. *Ontology matching*. Springer, Heidelberg (DE), 2007.

[17] J. Euzenat, H. Stuckenschmidt, and M. Yatskevitch. Introdution to the ontology alignment evaluation 2005. In *Proceeding of Intergrating Ontologies workshop at K-CAPO 2005*, 2005.

[18] F.McNeill-P. Shvaiko J.Pane F. Giunchiglia, M.Yatskevich and P.Besana. Approximate structure preserving semantic matching. In *ECAI 2008*, 2008.

[19] M. R. Genesereth and R. E. Fikes. Knowledge interchange format, version 3.0 reference manual. Technical report, Computer Science Department, Stanford University, 1992.

[20] F. Giunchiglia. Contextual reasoning. Technical report, IRST, Istituto per la Ricerca Scientifica e Tecnologica, 1992.

[21] F. Giunchiglia, F. McNeill, M. Yatskevich, J. Pane, P. Besana, and P. Shvaiko. Approximate structure preserving semantic matching. In *On the Move to Meaningful Internet Systems: OTM 2008*, pages 1217–1234, 2008.

[22] F. Giunchiglia, F. McNeill, M. Yatskevich, C. Sierra, and J. Sabater. Evaluating good answers in open knowledge. Technical report, OpenKnowledge, 2007.

[23] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-match: an algorithm and an implementation of semantic match. In *Proceeding of the European Semantic Web Symposium*, pages 61–75, 2004.

[24] F. Giunchiglia, M. Yatskevich, and E. Giunchiglia. Efficient semantic matching. In *ESWC'05*, pages 272–289, 2005.

[25] F. Giunchiglia, M. Yatskevich, and F. McNeill. Structure preserving semantic matching. In *Proceedings of the ISWC+ASWC International workshop on Ontology Matching (OM)*, Busan (KR), 2007.

[26] P. Grenon and B. Smith. Snap and span: Towards dynamic spatial ontology. *Spatial cognition and computation*, 4(1):69–104, 2004.

[27] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[28] T. R. Gruber and G. R. Olsen. An ontology for engineering mathematics. In *KR*, pages 258–269, 1994.

[29] M. Gruninger and M. Fox. The logic of enterprise modelling. 1995.

[30] Li Guo, D. Robertson, and J. Chen-Burger. A novel approach for enacting the distributed business workflows using bpel4ws on the multi-agent platform. In *IEEE Conference on E-Business Engineering*, pages 657–664, 2005.

[31] A. Hameed, A. Preece, and D. Sleeman. *Ontology Reconciliation*, pages 231–250. Springer Verlag, Germany, 02 2003.

[32] M.F. Hassan, D. Robertson, and C. Walton. Addressing constraint failure in agent interaction protocol. In *Proceedings of the 8th Pacific-Rim International Workshop on Multi-Agents (PRIMA '05)*, 2005.

[33] Y. Kalfoglou and M. Schorlemmer. Ontology mapping: the state of the art. *The Knowledge Engineering Review*, 18(1):1–31, 2003.

[34] N. Kavantzas, D. Burdett, G. Ritzinge, T. Fletcher, Y. Lafon, and C. Barreto. Web services choreography description language version 1.0. http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/, November 2005.

[35] D. Lambert and D. Robertson. Matchmaking multi-party interactions using historical performance data. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-05)*, pages 611–617, 2005.

[36] D. Lenat and R. Guha. *Building Large Knowledge-Based Systems*. Addison Wesley, 1990.

[37] C. D. Manning and H. Schutze. *Foundations of statistical natural language processing*. MIT Press, 1999.

[38] J. McGinnis and D. Robertson. Realizing agent dialogues with distributed protocols. 2004.

[39] F. McNeill, A. Bundy, and M. Schorlemmer. Dynamic ontology refinement. The University of Edinburgh, 2003.

[40] G. A. Miller. Wordnet: a lexical database for english. *Commun. ACM*, 38(11):39–41, 1995.

[41] I. Niles and A. Pease. Towards a standard upper ontology. In *FOIS '01: Proceedings of the international conference on Formal Ontology in Information Systems*, pages 2–9, New York, NY, USA, 2001. ACM Press.

[42] I. Niles and A. Pease. Towards a standard upper ontology. pages 2–9, 2001.

[43] N. Noy and M. Klein. Ontology evolution: Not the same as schema evolution. *Knowledge and Information Systems*, 6:428–440, 2004.

[44] S. Nuno and J. Rocha. Mafra - an ontology mapping framework for the semantic web. In *Proc. of the 13th European Conf. on Knowledge*, 1999.

[45] N. Osman, D. Robertson, and C. Walton. Run-time model checking of interaction and deontic models for multi-agent systems. In *Proceedings of the Third*

European Workshop on Multi-Agent Systems, pages 248–259, Brussels, Belgium, December 2005.

[46] M. Paolucci, T. Kawmura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Semantic Web - ISWC2002*, 2002.

[47] J. Pasley. How bpel and soa are changing web services development. *Internet Computing, IEEE*, 9, issue 3:60– 67, May-June 2005.

[48] F. Puhlmann and M. Weske. Using the pi-calculus for formalizing workflow patterns. In *3rd International Conference, BPM 2005*, volume 3649/2005, pages 153–168. Springer, 2005.

[49] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal: Very Large Data Bases*, 10(4):334–350, 2001.

[50] N. Reithinger, R. Engel, M. Kipp, and M. Klesen. Predicting dialogue acts for a speech-to-speech translation system. In *Proc. ICSLP '96*, volume 2, pages 654–657, 1996.

[51] D. Robertson. A lightweight coordination calculus for agent systems. In *Declarative Agent Languages and Technologies*, pages 183–197, 2004.

[52] D. Robertson. Multi-agent coordination as distributed logic programming. In *International Conference on Logic Programming*, Sant-Malo, France, 2004.

[53] D Robertson, C Walton, A Barker, P Besana, Y Chen-Burger, F Hassan, D Lambert, G Li, J McGinnis, N Osman, A Bundy, F McNeill, F van Harmelen, C Sierra, and Giunchiglia F. Models of interaction as a grounding for peer to peer knowledge sharing. *Advances in Web Semantics I: Ontologies, Web Services and Applied Semantic Web*, 4891/2009:81–129, 2009.

[54] J.R. Searle. *Speech acts: an essay in the philosophy of language*. Cambridge University Press, 1969.

[55] M. Shaw and B. Gaines. Comparing conceptual structures: Consensus, conflict, correspondence and contrast, 1989.

[56] P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *Journal on Data Semantics*, 4:146–171, 2005.

[57] C. Sierra, R.J. Aguilar, P. Noriega, J. Arcos, and M. Esteva. Engineering multi-agent systems as electronic institutions. *European Journal for the Informatics Professional*, 4, 2004.

[58] M. Smith, C. Welty, and D. McGuinness. Owl web ontology guide, raccomandation. web, February 2004.

[59] R. Studer, V. Benjamins, and D Fensel. Knowledge engineering: Principles and methods. *Data and Knowledge Engineering*, 25:161–197, 1998.

[60] Willis J. Tilley CB. Unified medical language system basics. National Library of Medicine, 2004. course presentation.

[61] M. Uschold, M. King, S. Moralee, and Y. Zorgios. The enterprise ontology. *Knowledge Engineering Review*, 13, 1998.

[62] W.M.P. van der Aalst and A.H.M. ter Hofstede. Yawl: Yet another workflow language. *Information Systems*, 30(4):245–275, 2005.

[63] R. van Eijk, F. de Boer, W. van de Hoek, and J.J. Meyer. On dynamically generated ontology translators in agent communication. *Internation Journal of Intellignet Systems*, 16:587–607, 2001.

[64] W.M.P. var der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. Technical report, http://www.workflowpatterns.com/, 2001.

[65] P. R. S. Visser, D. M. Jones, T. J. M. Bench-Capon, and M. J. R. Shave. An analysis of ontological mismatches: Heterogeneity versus interoperability. In *AAAI 1997 Spring Symposium on Ontological Engineering*, Stanford, USA, 1997.

[66] G. Wiederhold. Mediators in the architecture of future information systems. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 185–196. Morgan Kaufmann, San Francisco, CA, USA, 1997.

[67] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley and Sons, 2002.

[68] R. Yager. *Advances in the Dempster-Shafer Theory of Evidence*. John Wiley, New York, 1994.