Computing and Informatics, Vol. 34, 2015, 210–232

A MULTI-LAYERED ARCHITECTURE FOR COLLABORATIVE AND DECENTRALIZED CONSEQUENCE FINDING[*]

Philippe Chatalic, Andre de Amorim Fonseca

Laboratoire de Recherche en Informatique (L.R.I.)
Université Paris-Sud/INRIA Saclay – Ile-de-France
Bat. 650
91405 Orsay Cedex
France
e-mail: chatalic@lri.fr, andre.amorimfonseca@gmail.fr

Abstract. The consequence finding problem consists in producing all the consequences of a logical theory or, depending on the application context, in a restricted subset of these consequences. When the available knowledge is naturally scattered among different sources of information, computing such consequences with respect to the global theory in a decentralized way is a challenging problem. This paper presents Somewhere2, a multilayered architecture that may be used to solve such consequence finding problems in peer-to-peer networks of collaborating entities, that may evolve over time. The general layout of this architecture is described as well as the roles of its main components. Thanks to a careful and modular design, the resulting framework is very generic. This facilitates alternative implementations of specific components as well as its extension with additional features. First experimental results are presented, illustrating the scalability and robustness of this architecture. This framework may be used as a robust building block for more advanced distributed applications, such as Peer Data Management Systems.

Keywords: Automated reasoning, collaborative problem solving, peer-to-peer architectures, inference systems, consequence finding

Mathematics Subject Classification 2010: 68T15, 68T35, 03B05

[*] This paper is a revised and extended version of a paper presented in [9]

1 INTRODUCTION

The *consequence finding* problem [20, 21] amounts to finding formulas that are consequences of a logical theory. Many practical applications involve reasoning tasks that aim at discovering such consequences, that are not explicit in the original theory. In many cases, not all consequences are sought, but only a restricted subset of these, according to some syntactical property, called a *production field* [29].

Consequence finding is more complex than the simple *proof finding* problem, for which a user simply wants to *verify* whether a formula is entailed or not by a theory. Besides, the latter may be viewed as a particular case of the former, since proving a formula by refutation can be solved by adding the negation of this formula to the theory and considering the production field reduced to the contradiction. Consequence finding has proved to be useful for wide range of problems involving diagnosis, abductive reasoning, hypothetical and non-monotonic reasoning, query rewriting as well as knowledge compilation (see [21] for a survey). While it has been addressed essentially in the centralized case, in this work, we consider distributed approaches of this problem.

A frequent motivation for distributed approaches is to exploit possible parallelism in order to speed up some computing task. The goal is then to decompose a complex task and to distribute the work among set of available ressources. As an example, several techniques have been proposed to allow parallel evaluation of logical queries in datalog programs (e.g. [26, 5]). Decompositions associated to distributed processing can also be a key for solving problems that are too hard to be solved in a centralized way. For consequence finding, this can typically be the case for large theories. The problem can rapidly become out of scope for the processing capabilities of a single computing unit. In such a case, one possible approach is to exploit structural properties of the original theory in order to decompose this theory into subparts. This approach has been followed in the context of theorem proving by [3, 4] and recently extended to the case of consequence finding in [7].

But the need for a distributed approach becomes essential in situations where the knowledge from which conclusions have to be drawn is intrinsically scattered at different places. In this paper we focus particularly on such application contexts.

This is typically the case in multi-agent systems, where each agent has its own knowledge/point of view and may have to collaborate with other agents in order to achieve its goals. This is also the case in some semantic data management systems. The goal is then to exploit simultaneously the content of multiple (often heterogeneous) sources of information, where each source describes the organization of its data using its own ontology. Query answering over such networked set of data sources requires reasoning capabilities over the relevant ontologies. It generally proceeds in two steps, the first of which is a query rewriting step, where the original query is reformulated, in terms of the languages of the different relevant ontologies. The obtained rewritings are then evaluated on the appropriate sources. For such systems, the query rewriting step can often be reformulated in terms of a consequence finding problem.

Given the ever growing number of information sources available over the web, peer-to-peer architectures look particularly promising for that purpose. The absence of any centralized control or hierarchical organization and the fact that in such systems each peer plays the same role gives much flexibility to accommodate to the dynamic nature of such networks. This also clearly contributes to the scalability and the robustness of such approaches. Such principles are at the core of Peer Data Management Systems (PDMS) such as EDUTELLA [23], PIAZZA [13] or SOMEWHERE [2].

The SOMEWHERE platform is based on a propositional Peer-to-Peer Inference System (P2PIS). It can be used as a stand-alone application for solving consequence finding problems over a set of distributed propositional theories. But it can also be abstracted into more complex frameworks requiring distributed reasoning over a network of peer theories expressed in richer languages, as far this knowledge can be translated in an equivalent way at the propositional level. This encompass PDMS, when peer ontologies are expressed using adequate restrictions of first order logic, as for instance RDFS, the first standard adopted by the W3C for the semantic web. It has also been shown in [1] that this is also possible for ontologies expressed using the description logic DL-LITE$_R$, one of the maximal fragments of the DL-LITE family, for which query answering over large amounts of data remains tractable [8]. Focusing at the propositional level for the reasoning allows efficient implementations of consequence finding algorithms. This approach has already proved to be successful in theorem proving. SAT solvers have gained during the last two decades several orders of magnitude in the performance. As a consequence, several problems are currently best solved by prior propositional reduction and the use of a SAT solver. For instance, current Answer Set Programming (ASP) solvers typically proceed in that way [11].

SOMEWHERE pioneering work proposed the first ever implemented version of a fully decentralized consequence finder. Its good scalability on fairly large networks of peers [2] has been very encouraging. Its code has served as a basis for further extensions, e.g. to offer more convenient languages to describe peers knowledge or to allow sound reasoning despite possible inconsistencies between the peers of the network [10]. However this system was rather a *proof of a concept* than a rock solid piece of code. Its code suffered from many flaws, inducing rather high maintenance costs and making its extensions quite complicated in practice. Moreover, it was difficult to deploy in a simple way and it lacked a number of essential features to make it convenient to use for real applications. At some time, all these reasons have motivated a complete reengineering of the whole system, in order to improve both its design, robustness and extensibility.

This paper presents the architecture of this new system SOMEWHERE2. In the next section we first recall the basic principles of P2PIS and consequence finding, as well as the approach followed to compute such consequences in a fully decentralized way. In Section 3, we describe the architecture SOMEWHERE2, its organization, the roles of its main components, its features and new functionalities. In Section 4 we present some experimental results and Section 5 discusses the related work.

2 CONSEQUENCE FINDING IN P2P INFERENCE SYSTEMS

The general setting of this work concerns peer-to-peer networks, where each peer can model its own knowledge using a local propositional theory and can solve inference tasks such as consequence finding, using its own knowledge and/or by interacting with its neighbors in the network. For the sake of self-containedness, we recall from [2] (with slight variations in the presentation) the formal characterization of the problem that we consider.

2.1 P2P Inference Systems

The P2P Inference Systems that we consider correspond to networks of propositional clausal theories $\mathcal{P} = \{P_i\}_{i=1..n}$. We assume that each peer has a unique identifier i and a proper *vocabulary* V_i. The *global vocabulary* of $\mathcal{P}$ is defined as $V = \cup_{i=1..n} V_i$. As usual, a *literal* l is either a symbol of $v \in V$ or its negation $\neg v$ and its opposite literal $\bar{l}$ is defined by $\bar{v} = \neg v$ and $\overline{\neg v} = v$. A *clause* is a disjunction of literals. We denote by $\mathcal{L}_V$ (resp. $\mathcal{L}_{V_i}$) the language of clauses without duplicated literals over V (resp. V_i) and we assume that each peer theory P_i is a set of clauses of $\mathcal{L}_V$. In the following, for the sake of simplicity, we use the peer identifier i as an index to denote that a propositional variable x_i that belongs to the vocabulary V_i.

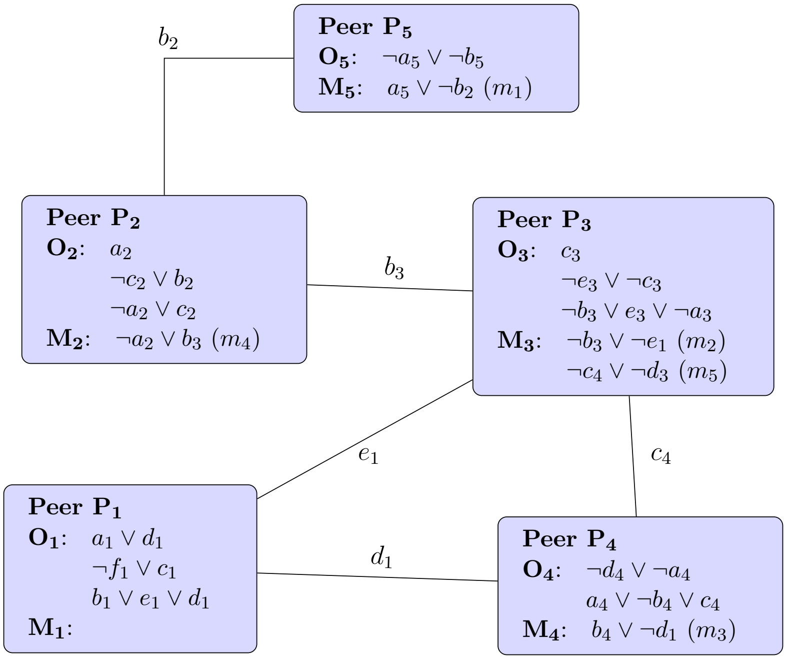
Definition 1 (Local clauses, mappings, shared variables and literals)**.** In a peer theory P_i, we distinguish *local clauses*, that are clauses containing only literals over the vocabulary V_i, from *mappings* that are clauses containing literals from the vocabularies of at least two different peers.

A variable of V_i is said to be *shared*, if it appears in a literal l of clause of P_i and if its opposite literal $\bar{l}$ appears in a mapping of another peer. In such a case both l and $\bar{l}$ are said to be *shared literals.*

Intuitively, local clauses describe the very own knowledge of a peer while mappings state logical constraints between the knowledge of different peers. Mappings characterize possible interactions between peers, through shared variables.

Shared variables implicitly define an *acquaintance graph*, the vertices of which correspond to the peers and whose edges link together peers containing opposite shared literals. Figure 1 is an abstract example of such a P2PIS acquaintance graph, the edges of which are labelled by the corresponding shared variables. The set of mappings and local clauses of a peer P_i are respectively denoted by M_i and O_i. The *global theory* of $\mathcal{P}$ is $\Sigma = \bigcup_{i=1..n} P_i = O \cup M$ where $O = \bigcup_{i=1..n} O_i$ and $M = \bigcup_{i=1..n} M_i$.

We assume that each peer P_i is aware of its shared literals and of its acquaintances, and given a shared literal l we denote by $\textsc{acq}(l, P_i)$ the set of other peers with some clause containing the opposite literal $\bar{l}$. For a clause c, we also denote by $S(c)$ the disjunction of its shared literals and by $L(c)$ the disjunction of the other literals of c.



Peer $\mathbf{P_5}$
$\mathbf{O_5}$: $\neg a_5 \vee \neg b_5$
$\mathbf{M_5}$: $a_5 \vee \neg b_2$ (m_1)
b_2
Peer $\mathbf{P_2}$
$\mathbf{O_2}$: a_2
$\neg c_2 \vee b_2$
$\neg a_2 \vee c_2$
$\mathbf{M_2}$: $\neg a_2 \vee b_3$ (m_4)
b_3
Peer $\mathbf{P_3}$
$\mathbf{O_3}$: c_3
$\neg e_3 \vee \neg c_3$
$\neg b_3 \vee e_3 \vee \neg a_3$
$\mathbf{M_3}$: $\neg b_3 \vee \neg e_1$ (m_2)
$\neg c_4 \vee \neg d_3$ (m_5)
e_1
c_4
Peer $\mathbf{P_1}$
$\mathbf{O_1}$: $a_1 \vee d_1$
$\neg f_1 \vee c_1$
$b_1 \vee e_1 \vee d_1$
$\mathbf{M_1}$:
d_1
Peer $\mathbf{P_4}$
$\mathbf{O_4}$: $\neg d_4 \vee \neg a_4$
$a_4 \vee \neg b_4 \vee c_4$
$\mathbf{M_4}$: $b_4 \vee \neg d_1$ (m_3)

Figure 1. A P2PIS network

2.2 Semantics of a P2PIS

The semantics that we consider for a P2P Inference System $\mathcal{P} = \{P_i\}_{i=1..n}$, is the classical semantics of propositional logic for the global theory $\Sigma = \cup_{i=1..n} P_i$.

Definition 2 (Interpretations, Models)**.** Let Σ be a set of clauses over of finite vocabulary V,

- an *interpretation* I of Σ is an assignment of each variable of V to true or false.
- for $v \in V$, the literal v (resp. $\neg v$) is *satisfied* by an interpretation I iff $I(v) =$ true (resp. false) and a clause c is satisfied by I iff one of its literals is satisfied by I.
- an interpretation I is said to be a *model* of a clause c iff I satisfies c (denoted by $I \models c$) and it is a model of Σ iff it satisfies each clause of $c \in \Sigma$.
- Σ is *consistent* iff it has a model.

The *logical consequence* relation $\models$ is classically defined by $\Sigma \models c$ iff any model of Σ is also a model of c. An interesting problem is then to characterize the *logical content* of a given consistent theory. More precisely, it is generally sufficient to characterize its strongest consequences, which corresponds to the notion prime implicates.

Definition 3 (Implicates, Prime Implicates)**.** Let Σ be clausal theory and c be a clause,

- c is an *implicate* of Σ iff $\Sigma \models c$,
- c is an *prime implicate* of Σ iff
  - c is an *implicate* of Σ
  - for any other implicate c' of Σ, if $c' \models c$ then $c \equiv c'$.

Because the number of (prime) consequents of a theory can be huge, producing the whole set of consequences is not necessarily the most important. It is often the case that one is rather interested in a particular subset of clauses $PF \subseteq \mathcal{L}_V$, called the *production field.* Most of the time such a set is not described in an extensional way but is rather characterized by some syntactical property that must be satisfied and that may vary according to the application context. Typical properties can require a maximal length for the clauses of interest, that these clauses are built on a restricted subset of literals or any combination of such requirements. The goal is then to determine the strongest consequences that belong to the production field PF, called the *characteristic clauses.* In the following, although we do not introduce special notations for that purpose, we assume the existence of such a production field and that our concern is to compute consequents that belong to this production field. The case where no restriction occurs simply corresponds to the case where $PF = \mathcal{L}_V$.

An interesting variant of this problem is to determine the influence of a new piece of information with respect to the set of characteristic clauses of a theory. This can be formalized by the notion of *proper* prime implicate.

Definition 4 (Proper prime implicate of a clause w.r.t. a clausal theory)**.** Let Σ be a clausal theory and c be a clause

- a clause c' is said to be an *implicate* of c w.r.t. P iff $\Sigma \cup \{c\} \models c'$
- c' is a *prime implicate* of c w.r.t. Σ iff
  - c' is an implicate of c w.r.t. Σ
  - for any other clause c" implicate of c w.r.t. Σ, if $c" \models c$ then $c" \equiv c'$
- c' is a a *proper prime implicate* of c w.r.t. Σ iff
  - it is a prime implicate of q w.r.t. Σ
  - $\Sigma \not\models c'$.

2.3 Decentralized Consequence Finding

While several approaches have been proposed to compute sets of (proper) prime implicates in a centralized setting [21], this is not the case for the distributed case. In a P2PIS setting, the problem addressed here is to compute the set of proper prime consequents of some clause c, typically stated using the language of some queried peer, with respect the global theory $\Sigma = \cup_{i=1..n} P_i$. But the major difficulty is that this theory is unknown. None of the peer in the P2PIS has a global view of the network. A peer only knows its own theory P_i and the variables it shares with its neighbours.

DeCA [2] is the first algorithm that has been proposed to solve this problem in fully decentralized way. Basically, it proceeds using a split/recombination strategy. Although implemented as an anytime message passing algorithm, it can also be described in a functional way. It has been established in [2] that both versions (functional and message passing) are sound and complete for the problem of computing the set of proper prime consequents and that they compute the same set of consequents.

We recall in Algorithm 1 the functional version of DeCA (in a slightly generalized version) and describe the main steps of this algorithm. We assume that the same algorithm is running in the same way on all the peers of the P2PIS and that DeCA(q, P, PF) corresponds to the case where the peer P is asked to compute the proper prime implicates of a clause q w.r.t. to the global theory of the P2PIS, for the production field PF. We also assume that PF is a *stable* production field, i.e. that any strict sub-clause of a clause of PF also belongs to PF.

Algorithm 1: Decentralized Consequence Finding Algorithm

```
DeCA($q, P, PF$)
(1)DeCAH($q, \{P\}, PF, \emptyset$)

DeCAH($q, SP, PF, hist$)
(1)  if for every $P \in SP$,  $(q, P, \_) \in hist$
(2)    return $\emptyset$
(3)  else if $(\bar{q}, \_, \_) \in hist$
(4)    return $\{\square\}$
(5)  else
(6)    RESULT $\leftarrow \emptyset$
(7)    foreach  $P \in SP$
(8)      LOCAL($P$) $\leftarrow \{q\} \cup LocalCF(q, P, PF)$
(9)    foreach  $P \in SP$ and $c \in$ LOCAL($P$)
(10)     if $c \in \cup PF$
(11)       RESULT $\leftarrow$ RESULT $\cup \{c\}$
(12)     if $S(c) \neq \square$
(13)       foreach literal $l \in S(c)$
(14)          Answer($l$) $\leftarrow$
(15)         DeCAH($l$, ACQ($q, P$), $PF, [(q, P, c)|hist]$)
(16)       if for every $l \in S(c)$ Answer($l$) $\neq \emptyset$
(17)         unionComb $\leftarrow \{L(c)\} \otimes (\otimes_{l \in S(c)}$Answer($l$))
(18)         RESULT $\leftarrow$ RESULT $\cup$ (unionComb $\cap PF$)
(19)   return RESULT
```

When a peer P is asked to compute the proper prime implicates of a literal q, the initial call just performs a call to DeCAH, which describes the simultaneous execution of DeCA on a set of peers SP. For the initial call we have $SP = \{P\}$. The additional parameter *hist*, initialized to $\emptyset$, is used as a structure to record the history of successive recursive calls to DeCAH. It is essentially used to avoid loops and useless calls in the reasoning (lines 1-4). This parameter is essential because no further assumption is made on either the topology of the P2P network (that may contain cycles) or the structure of the peer theories.

Then for each peer $P \in SP$, we first compute all proper prime implicates of q w.r.t. its own local theory and the production field. This is the role of $LocalCF(q, P, PF)$ (line 8) which only keeps those consequents c for which $L(c) \in PF$ (it can be shown that those that do not satisfy this property cannot contribute to produce a consequent in PF). Among the consequents produced so far, those that match the production field are immediately added to the result to be returned. Moreover, any clause c that contains shared literals is split in two subclauses: $L(c)$ and $S(c)$. $S(c)$ is then split in turn and for each shared literal l of $S(c)$ DeCA asks its relevant neighbors in $\textsc{acq}(l, P)$ (which are running the very same algorithm) to compute similarly the proper consequences of l w.r.t. the P2PIS. When the answers for all literals of $S(c)$ have been obtained they are recombined together (using the distribution operator $\circledvee$[1]) as well as with the non shared literals $L(c)$, to produce the new consequences. Clauses obtained in this way and that belong to the production field PF are then also returned as answers for the initial query q.

The main differences between the functional and message passing versions of DeCA is that the latter is an anytime algorithm, that uses asynchronous communications between peers. Moreover, answers are returned and recombined in an incremental way. The reader is referred to [2] for further details.

2.4 Illustrative Example

In order to get a better understanding of the DeCA algorithm, let us illustrate its behavior on the network of Figure 1, when computing the proper prime implicates of d_4 on P_4. For simplicity, we assume here that the production field $PF = \mathcal{L}_V$ (i.e. the set of all possible clauses).

- Local consequents of d_4 on P_4 are first computed, which gives: $\{d_4, \neg a_4, \neg b_4 \vee c_4, \neg d_1 \vee c_4\}$. These clauses are immediately returned as answers for the initial query. But since c_4 and $\neg d_1$ are shared literals, this also triggers two new queries, to compute the (proper prime) consequents of c_4 on P_3 and of $\neg d_1$ on P_1.
- On P_3, the local consequents of c_4 are $\{c_4, \neg d_3\}$. Since d_3 is not shared, the reasoning stops there and both answers are returned to P_4, as answers for the query c_4 (where they will have to be recombined with the non-shared literal $\neg b_4$).

[1] If S_1 and S_2 are sets of clauses, $S_1 \circledvee S_2$ is defined as $\{c_1 \vee c_2 \,|, c_1 \in S_1 \text{ and } c_2 \in S_2\}$.

- On P_1 the set of local consequents of $\neg d_1$ is $\{\neg d_1, a_1, b_1 \vee e_1\}$. These clauses are sent to P_4, as answers for the query $\neg d_1$. But since e_1 is a shared literal, this also triggers a new query e_1 for P_3.
- On P_3 the set of local consequents of e_1 is $\{e_1, \neg b_3\}$. These clauses are sent to P_1, as answers for the query e_1. But since b_3 is a shared literal, with P_2, this triggers a new query $\neg b_3$ for P_2.
- On P_2 the set of local consequents of $\neg b_3$ is $\{\Box\}$ (which subsumes all other clauses). $\Box$ is returned as an answer to P_3, where it subsumes e_1 and $\neg b_3$. Eventually P_3 returns $\Box$ to P_1, as an answer to the query e_1
- On P_1, $\Box$ (which subsumes e_1) is then recombined with b_1 and the set of consequents of d_1 becomes $\{\neg d_1, a_1, b_1\}$. This set is returned to P_4.
- On P_4, these three clauses are now recombined in place of $\neg d_1$ in the clause $\neg d_1 \vee c_4$ with the answers obtained of c_4, producing: $\{\neg d_1 \vee c_4$, $a_1 \vee c_4$, $b_1 \vee c_4$, $\neg d_1 \vee \neg d_3$, $a_1 \vee \neg d_3$, $b_1 \vee \neg d_3\}$. These answers are added to those previously obtained, namely $\{d_4, \neg a_4, \neg b_4 \vee c_4, \neg b_4 \vee \neg d_3\}$.

Eventually, the set of proper prime consequents of d_4 w.r.t. to the global theory returned by the DeCA algorithm is:

$$\left\{ \begin{array}{lll} a_1 \vee c_4, & a_1 \vee \neg d_3, & d_4, \neg a_4, \\ b_1 \vee c_4, & b_1 \vee \neg d_3, & \neg b_4 \vee c_4, \\ \neg d_1 \vee c_4, & \neg d_1 \vee \neg d_3, & \neg b_4 \vee \neg d_3 \end{array} \right\}$$

As we can see on this example, the communication between peers induced by the DeCA algorithm, requires the use of different kinds of messages. The *query* messages are sent to ask a peer to compute the consequences of a clause, given some reasoning history. The *answer* messages are used to return (some) answers to peers asking for such queries. Since answers are returned in an incremental way, a third kind of message (*finish* messages) is also used to inform a peer that all possible answers to a given query have been produced and sent back to the querying peer.

2.5 Dealing with Possible Inconsistencies

The P2PIS considered in [2] are assumed to be consistent. However, in a fully decentralized network where each peer is autonomous and may freely set up its mappings, such an assumption, is not realistic. All peers do not necessarily agree on all topics, some of them may have done mistakes or some malicious peer could want to screw up the network. If the global theory is inconsistent, one may wonder about the soundness of the derived conclusions, since any formula is then a consequence of global theory.

This problem has been addressed in [10, 25], where it has been shown that it is also possible to detect inconsistencies in a P2PIS in a fully decentralized way. Moreover, it is possible as well to adapt the reasoning techniques used in DeCA to insure that the derived consequents can be produced from sound subsets of the global theory.

This approach assumes that at least each peer theory P_i is consistent, and that potential causes of inconsistencies are due to mappings. It describes an algorithm called P2P-NG for detecting *nogoods* (i.e., minimal sets of mappings that entail inconsistency). P2P-NG is called each time that a peer P_i wants to create a new mapping m. It checks if the empty clause $\square$ is a proper implicate of m. It proceeds in similarly way to DeCA using a production field $PF = \{\square\}$, except that it considers *all possible derivations* of $\square$ from m. For each of them it records its *mapping support* (i.e. the set of mappings used in the proof). Each such proof induces a new discovered nogood that is then stored on the peer P_i. As a consequence, all possible causes of inconsistencies are detected incrementally and stored in the P2PIS in a decentralized way.

Another algorithm WF-DeCA is then described in [10, 25], that allows to compute the so-called *well founded* proper consequences of an input clause w.r.t. to the global theory, while avoiding those that can only be produced from inconsistent subsets of the global theory. It uses the same split/recomposition strategy as DeCA, but also has to deal with the additional complexity induced by the management of multiple mapping supports and the collection of relevant nogoods.

From an implementation point of view, the code of P2P-NG and WF-DeCA presents many similarities but also significant differences with the original code of DeCA. Although it evolved from the same source, it was developed by different persons, with different coding practices and style as a result. As a consequence simultaneous maintenance of these different pieces of code has turned out to be very tricky and costly.

3 A MULTILAYERED ARCHITECTURE FOR SOMEWHERE2

3.1 Motivations and Objectives

The first experimental results obtained with the SOMEWHERE platform have been very encouraging, in particular from its scalability perspective. Nevertheless, this system was rather a *proof of concept*, offering minimal services for the purpose of the aimed experiments. It suffered from a number of limitations and lacked a number of essential features to make it convenient to reuse in real applications.

One of the major limitation of this framework was the impossibility to deal with really dynamic networks. All peers had to be configured in advance (i.e. before being launched) and the peer addressing schema used for the communications between peers required fixed addresses to be encoded into peer theories. As a consequence, it was impossible for a new peer to join an existing network of peers. And the peer communication protocol was not explicitly handling the possibility for a peer to leave the P2PIS, what caused lots of instabilities in the case of a faulty process. This also made the effective deployment of such P2PIS on different physical networks particularly cumbersome. Moreover, the communication protocol was rather limited and implemented using socket-level interactions, that was not able to pass beyond firewalls.

Another limitation is that it was impossible to modify the peer theories. Functionalities from the user interaction point of view where reduced almost to the minimum, i.e. the possibility for a user to ask a peer for computing the proper prime consequents of a formula and, for the inconsistency-tolerant variant, to add a new mapping and detect possible nogoods. But the structure of the code did not facilitate the easy addition of such new commands.

From a software engineering point of view, the original code, as well as the various extensions derived from it, suffered from many flaws. There was a lot of duplicated code. The development did not followed strict naming/visibility rules and homogeneous coding practices. The granularity of some pieces of code was also unequal. And there were lots of serious cross cutting concerns that strongly affected the modularity of the whole. The lack of modularity was really a crucial problem, not only because this made the code maintenance tasks more complicated but also because it was very difficult to test and compare alternatives implementations of the same functionalities. For instance, the consequence finder used to produce the local consequents on a peer followed a rather naive strategy. But it was difficult to abstract this component from the rest of the code, to provide a more efficient procedure. Another weakness was the lack of sufficient unit and integration tests. As a consequence, it was difficult to ensure that some attempts to improve/extend the code preserved all expected behaviors.

The need for reconsidering the whole architecture arose from all these facts. In order to really improve the structure, robustness and flexibility of the new framework, several objectives have been targeted, among which are the will to:

- reinforce the modularity of the code as much as possible
- make it easier to use alternative encoding of a same functionality
- add a real and extensible user interface
- introduce an abstract layer to characterize the communication protocol and facilitate the extensibility of the later
- rely on a tested peer-to-peer framework for managing the network of peers and concrete communications between peers
- improve the performances of the local consequence finding algorithm
- factorize as much as possible the common code between the different variants

Moreover, this was also an opportunity to introduce better software engineering practices in the development cycle and use appropriate tools for source code management and versioning, as well as for automated testing.

3.2 Architecture General Layout

Improved robustness has been obtained by a careful analysis of the functional dependencies between the different parts of the code. The goal was to reduce such dependencies as much as possible. This has lead to the design of a multilayered architecture, organized around several *components*. Each component itself is organized in several layers. Essential functionalities are first described at an abstract level and then implemented at a particular level. Such a multi layered approach greatly improves a flexibility of the code. Particularly, alternative implementations for the same component can be introduced and tested without altering the existing code.

The flexibility of SOMEWHERE2 also results from the organization of the code which is structured around the abstract notion of a *module*. Each module addresses a specific concern. While some *required modules* are automatically loaded by the application, others can be included (or not) at build time, according to the user's needs. A *module manager* is responsible for loading and configuring the different modules in an appropriate way. The genericity of this approach also improves the flexibility when it comes down to creating extensions and/or selecting different functionalities during compilation.

The general layout of SOMEWHERE2 architecture is described in Figure 2. It is made up of 4 components: *Module Manager*, *User Interface*, *Transport* and *Distributed Reasoning*, each of which consists of several modules. The default configuration of SOMEWHERE2 also rely on an extra component called *IASI Libs*, that offers reasoning services, but in a centralized setting. Since such a library can also make sense for other applications independent of *Somewhere2*, it is rather seen here as an external dependency.

The different components have very little dependencies. These are represented by top-bottom adjacencies. For example: the *Distributed Reasoning* component only depends on the *Transport* and *IASI Libs* components. The *Transport* component, on the other hand, only depends on the *Module Manager* component. Dependencies between modules inside a same component are reflected in the same way.

Each component can contain required, abstract, concrete and/or external modules. The `baseApp` module plays here a central role, since it is responsible for the instantiation and configuration of the other modules. Each module may in turn have specific libraries dependencies and its own configuration scheme. The `swr.cli` module extends the `swr.baseApp` module. It corresponds to the application entry-point and provides an interface for the management of command line arguments. A generic model of commands has been introduced to make the extension with new commands very easy. The format of peer theories descriptions has also been enhanced in order to facilitate the deployment on physical networks versions.

3.3 The Roles of the Main Components

We briefly describe noticeable features of some of these components.

User Interface This component is responsible for all interactions with a running peer. It models a generic notion of command in the abtract module (*ui*) that may be easily used to provide new interaction possibilities. Currently, there

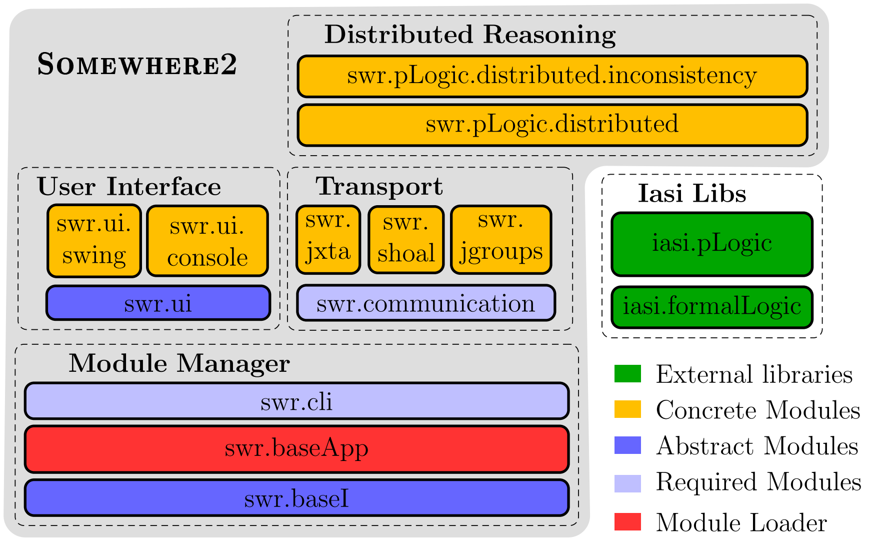


Figure 2. SOMEWHERE2 architectural schema

is only one concrete module (*ui.console*) that corresponds to a command line interface. But as illustrated in Figure 2, this is the place where the application could be extended with graphical user interface (e.g. *ui.swing*).

Transport In contrast with the previous version of SOMEWHERE, where all communications were handled in an ad hoc manner at the socket level, the new architecture has been designed in order to allow reusing an existing P2P framework. The abstract *communication* module describes the concurrency model (which is based on http-like sessions). It defines the core concepts required by the application for exchanges between peers (messages types, processors, sessions) and all what is relative to the evolution of the network (joining/leaving peers, lost connections, . . .).

This abstract layer has really proved to be precious since during the development. Due to instabilities and/or obsolescence of some different P2P frameworks, three different concrete modules have been implemented, corresponding respectively to `jxta` [17], `shoal` [28] and `jgroups` [16], without affecting other part of the code.

Distributed Reasoning This component is responsible for all knowledge level aspects, relevant to the distributed consequent finding algorithms described in [2, 10]. It defines the type of messages exchanged between peers (at the application level) and those corresponding to relevant user interface commands (e.g. modification of a theory, addition of mappings, . . .), the message handlers for the queue, the anytime recombination algorithm. This component is structured in two modules. The module responsible for algorithms that tolerate inconsistency, implement the management of mapping supports and nogoods, as well as the specialized algorithms P2P-NG and WF-DeCA. This module depends on the *pLogic.distributed* module, and thus shares as much code as possible with the classical version of DeCA. Both modules still support some level of abstraction, that allows to easily implement (and thus to compare) different strategies for various key points of these algorithms.

IASI Libs Although it has its own interest (independent of Somewhere2) and is thus packaged as an independent project, this component has been developed simultaneously to the other modules. It is the core library that is used for performing local consequent finding by the peers. Here again, various abstraction level have been introduced, that allow using alternative implementations of common data structures and of different consequence-finding algorithms. In contrast with the original version the local consequence finder of which was based on a simple split/backward chaining strategy, the consequence finder used in Somewhere2 may be seen as a corrected and optimized version of IPIA [18, 19].

Besides, the complete reorganization and redesign of the global architecture, many improvements and functionalities have been added to the new framework. The most noticeable one being probably the real ability to cope with the dynamicity of networks, with peers that may safely join and leave the network. It is also possible for a peer to modify its theory.

The robustness of Somewhere2 is also the consequence of a permanent effort to follow good software engineering practices, such as the systematic use of unit tests, the intensive use of design patterns and of static code analyzers (Sonar). A Jenkins server has also been configured to set up integration tests. In its current state, the project corresponds to around 13 000 lines of Java code, with less than 5 % redundancy. It is structured as a set of Maven projects with dependencies to ease the build process.

4 FIRST EXPERIMENTAL RESULTS

In this section we present preliminary experimental results that have been obtained using Somewhere2. Our main objective here is not to present an exhaustive experimental study, that would require further work, but rather to show that the new platform behaves well and scales up in a satisfactory way even on hard instances.

From a methodological point of view, since an extensive experimental work has already been performed with the former platform, it seems natural to try performing similar experiments and to compare the results obtained with both architectures. However one should be careful with raw comparisons because these are meaningful only for the cases where the experimental conditions are exactly the same and where the tools that are compared compute the very same result, under the same assumptions. Yet, although Somewhere and Somewhere2 have both been designed to solve the same kind of problems, i.e. to produce sets of proper prime consequents, there are some significant differences in the design of the two architectures that can induce a serious bias on raw comparisons.

One major difference is that the original SOMEWHERE includes a *server* functionality, that allows hosting multiple peers on a single server. This feature was introduced in order to make it possible to perform experiments involving a large number of peers (up to 1 000) while a limited number of machines (75 for the experiments reported in [2]). On this platform, communications between peers hosted on a same server are managed directly by the server (thus avoiding any network traffic). In contrast, with the new architecture each peer has to be run as a separate instance of SOMEWHERE2 and communications between different peers necessarily transit through the network. Moreover, one may also expect some overhead induced by layered design on SOMEWHERE2 and particularly by the use of a separate framework for P2P communications, while the earlier platform operates directly at the socket level.

Another significant difference between the two approaches comes from the algorithms that are used to compute local consequences on each peer. The former approach uses a naive backward chaining strategy. It does not require the prior saturation of the local theory but can in some cases produce some consequents that are not proper ones. If these consequents contain shared literals, this can increase the number of queries asked to neighbor peers and thus increase the traffic volume. In contrast, the latter approach uses a more optimized consequence finder that ensures that all consequents produced are proper ones. From these two observations, it is clear that the point here cannot be to compare raw experimental results, but rather to verify that similar behaviors can be observed for similar instances.

As a consequence, in this study we have considered problem instances having the same characteristics as those studied in [2]. These correspond to random networks of peers generated using a *small world* graph generator (in the sense of Watts and Strograd [31, 24]), supposed to reflect properties encountered in social networks. The peer theories are then also generated in a random way, first by generating a set of m clauses of length 2 over a set of n propositional symbols. For each peer, a subset of t ($t \leq n$) *target variables* is also randomly generated. These symbols are used to define the production field PF as the set of clauses containing only literals made from these variables. Then, for each pair of connected peers in the small world graph that has been generated, we add a number of q mapping clauses of length 2, by randomly picking one literal over the vocabulary of each peer. And for some of them, we add a third literal, again randomly picked over the vocabulary of both peers (we denote by $p3m$ the proportion of *3-mappings* i.e. that have 3 literals). Each mapping is then randomly inserted in one of the two peer theories. We thus obtain local theories that correspond to $2+p3c$ *formulas*, i.e. sets of clauses containing only clauses of length 2 and 3 (where $p3c$ – which is different from $p3m$ – denotes the proportion of *3-clauses* in the whole theory). The characteristics of such random theories have been studied by Schrag and Crawford [27], for the problem of prime implicate computation, and this study has been extended in [2] to the case of proper prime implicates. It has been shown that even for very small values of

$p3c$ the number and total size of proper/prime implicates grows exponentially. As a consequence, reasoning on local theories can already be quite hard, which of course means that distributed reasoning on a large number of such theories can get even harder.

For this set of experiments, we have considered local theories with more or less the same size as those considered in [2], with $m = 70$ binary clauses and over a vocabulary of $n = 70$ variables and a subset of $t = 40$ target variables for the production field. We have the considered small world graphs of 100 peers, with an average number of 5 neighbors for each peer. Then for each pair of neighbors peers we have added $q = 5$ additional mapping clauses, with a varying proportion of 3-clauses. This represents 25 mappings between each pair of connected peers and, since these are randomly assigned to the two peers, an average of 82.5 clauses for local theories. The global theory thus contains 8 250 clauses over 700 variables and 400 target variables for the production field. Since the proportion of $p3m$ of mappings of length 3 is known to have strong impact on the difficulty of the problem, we have considered 4 instances, denoted in the following tables by *very easy*, *easy*, *medium* and *hard* corresponding to values for $p3m$ of 0, 20, 50 and 100 %, respectively. Note that, while the three first instances are in fact less easy than those with similar labeling in [2], the *hard* cases in both experiments have similar characteristics.

For each instance, we then have generated a set of 500 random queries (actually 1000 for the two first instances), corresponding to random literals over the global vocabulary (with an equal positive/negative probability). We then have deployed the P2PIS corresponding to the different instances on clusters of the Grid'5000[2] network. The machines used for these experiments were equipped with the quad core CPU Xeon Processor X3440 (64 bits, 2.53 GHz, 8 Mb cache memory) and 4 Gb of main memory, under a Linux OS. Each query has been launched on the appropriate peer, with a time limit of 60 s, after which a timeout event is triggered and the computation is interrupted. The table 1 accounts for average time observed to obtain answers with SOMEWHERE2.

The first four lines report respectively the average time (in seconds) required to obtain the first, 10th, 100th and 1 000th answer corresponding to the query (character '–' means that there where less answers than the corresponding number). We mention just below the percentage of the queries that effectively returned at least as many answers (e.g. for the second line and second column, 51.3 % of the queries asked to the peers returned at least 10 consequents, and the average time for producing the 10th answer for those queries was 0.123 s). The last two lines indicate respectively the average and median time over all answers (including those that reached the timeout limit). For information, we also recall in the last column the corresponding results obtained for the "hard" instance described in [10], since its characteristics are almost the same, apart from the fact that it was in a larger net-

[2] Grid'5000 is a French large grid project, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several universities as well as other funding bodies [12]

	Somewhere2				Somewhere
Difficulty p3m	Very Easy 0 %	Easy 20 %	Medium 50 %	Hard 100 %	Hard 100 %
1st ans.	0.029 100 %	0.030 100 %	0.023 93.3 %	0.080 89 %	1.39 89.3 %
10th	0.262 27.7 %	0.123 51.2 %	0.170 54.0 %	0.828 28.1 %	1.13 12.0 %
100th	– –	– –	0.276 8.4 %	2.420 12.1 %	4.09 10.7 %
1 000th	– –	– –	– –	– –	11.35 7.15 %
Average Median	0.021 0.020	0.211 0.023	1.612 0.113	1.411 0.093	N.A. N.A.

Table 1. Answer time (in seconds) over 500/1 000 queries

work of peers (unfortunately, average and medium time had not been recorded for these experiments).

The Table 2 reports the proportion of queries that have been completed within the time limit and of those that have been interrupted by a timeout event. Most of these timeouts are due to hard queries that were really too complex to be solved within the allotted amount of time (the production field that is used here is quite large). A few of them can also be explained by some grid instabilities.

	Somewhere2				Somewhere
Difficulty p3m	Very Easy 0 %	Easy 20 %	Medium 50 %	Hard 100 %	Hard 100 %
% success	99.37	99.28	94.13	**90.56**	33.1
% timeout	0.63	0.72	5.87	9.44	66.9

Table 2. Proportion of success/timeouts over 500/1000 queries

As adviced before, raw comparisons with the results obtained with the earlier version of Somewhere should be considered very cautiously, because of the previously mentioned differences between the two. Nevertheless, several points are worth noticing.

First of all, for the easiest cases, propagations in the network should be limited because the depth and width of the reasoning are known to be small [2]. As a consequence the reported time essentially reflects the effort spent on local computations. And since the theories are similar in size, this suggests a clear improvement of the performances of the local consequence finder. This is however not surprising since a significant effort has been put in improving this part of the code (in IASILibs).

A second observation is that for the most difficult instances, performances still remain in average much better than what could be observed with the earlier version (up to a factor 2 for the difficult case). This looks more surprising since, as the proportion of 3-clauses increases, the number of messages exchanged in the network should augment in a significant way. Since in such networks the communication time generally dominates local computation time, one could expect performances to decrease. This should result from the fact that prior savings due to the server functionality in SOMEWHERE do not occur anymore. Each peer runs here on a different node of the grid. Moreover, one could also expect some additional overhead induced by the introduction of the different layers in the new architecture. On the other hand, when comparing these results with those of [2], we observe that the number of returned answers is much smaller with the new version. Two possible explanations might explain such a behavior. This can be due to the new strategy used to compute local consequents, that filters out non proper prime implicates. But it could also be biased by the fact that the number of peers is here 10 times smaller than for the corresponding instances in [2]. Experiments on similar, but larger networks, will probably help to investigate further on this point.

The proportion of observed timeouts also vary in a significant way. As mentioned earlier, most of these correspond to queries that trigger a very large number of messages between neighbor peers. We here observe that for the *hard* case we have less than 10 % timeouts, while with the previous version this rate was around 67 %. One should however remain extremely cautious again with such a raw comparison. First, we here allowed 60 s per query while in previous experiments the time limit was set to 30 s. But this difference alone cannot explain such an important improvement. Another possible explanation could come from the differences between the local consequence finder used in both approaches. Since non proper prime implicates are filtered out, this reduces the number of subqueries to neighbors and thus raises the chances for the whole process to complete during the time limit. The improvement might also reveal that some situations (typically with communication failures) are better handled with the new architecture than with the former one. Indeed, a timeout occurs as soon as a single message is missing in the whole tree of induced subqueries. And again, the smaller size of the networks considered in this paper could also contribute to explain the reduction of timeouts.

The reason for not having considered much larger networks until now is that this requires using simultaneously a large number of nodes in the grid for a relatively long period of time. Although it is possible to request for such time slots with sufficient nodes, Grid'5000 resources are shared between many users for different kinds of experiments. Usage policies stipulate that such expensive experimentations should remain occasional. Therefore our goal was to experiment first with medium size networks before stepping to larger networks. Since these results look promising this is definitely one of the next steps in the agenda.

5 RELATED WORK

To the best of our knowledge, the first SOMEWHERE framework was until recently the only implemented distributed consequence finder and the previous sections have already stressed the main differences between the two platforms.

Recently it has been proposed in [7, 15] to generalize the ideas of theory partitioning presented in [3, 4], originally designed to solve theorem proving techniques, to the case of consequence finding. This method exploit a tree decomposition of the original theory into a set of buckets gathering formulas expressed on subsets or the original. Symbols common to different buckets constitute the *communication language* between these buckets. In the original approach, information is propagated from leaves of the tree to its root. The idea is to compute the consequences of the information gathered so far, in the communication language of the current bucket with its parent. While for the theorem proving approach, the ultimate goal is to derive a contradiction in the root bucket, the idea is to enlarge the set of consequents produced on each bucket to sets of clauses, corresponding to disjunctions of a subclause that belongs to the production field and of a subclause that may be expressed using the communication language. They propose adaptations of the SOLAR consequent finder [22] to this task and study different strategies (sequential and parallel) for exploiting such decompositions. One of the strength of this approach is that it can be used over instances expressed in first order language. Bounding strategies have however to be used on first order instances in order to avoid possible loops in the reasoning. This work presents experimental results on a number of various examples, either at the propositional of the first order level. Good results are apparently obtained in many cases but they seem quite sensitive to the tuning of several parameters. Anyway, although it is a distributed approach, it clearly presupposes to have the global knowledge available, which contrasts to our fully decentralized approach.

In another work [14] the same authors have tried to compare their previous approach with an alternative one, that does not presuppose any decomposition and thus is supposed to fit better to multi-agent like application contexts. They also describe a message passing algorithm for consequence finding in a very high level way. However, this approach looks highly non-deterministic and its description is lacking sufficient details to get a precise understanding of the behavior on the whole. It considers a case where the production field is the whole language. The propagation strategy apparently corresponds to a general saturation of all the peer theories according to their languages, but since these languages can evolve as new clauses are received, it is not clear whether this amounts or not to some form of centralization of the knowledge on most peers of the network, an approach that would not be much convincing.

Another extension of the DECA algorithm has been proposed by [6] for the case where the global knowledge of the P2PIS is inconsistent. Authors suggest to adapt reasoning techniques from argumentation theory, in order to arbitrate cases where different conflicting pieces of knowledge could be derived in a well founded way (but from different consistent subsets). It can be seen as more refined alternative to [10]. However, this approach has been evaluated through some simulation, but has not been implemented in a really decentralized framework.

6 CONCLUSION AND PERSPECTIVES

We have presented the architecture of SOMEWHERE2, a complete reengineering of the SOMEWHERE framework. Its main improvements in comparison with the previous framework are the following:

- the structure of the code has been completely reorganized as a set of components with minimal dependencies, thus eliminating most crosscutting concerns,
- most components expose their features at an abstract level in separate modules, thus allowing the easy integration of alternative implementations,
- the multilayered organisation clearly defines the interactions between components, this facilitates the integration of further extensions as well as the maintenance of the existing code,
- it reunifies in a single and coherent framework both variants, tolerant or not too inconsistent theories,
- it is now able to deal with network of peers which topology can evolve over time,
- peers can also safely modify their own theory, without compromising the stability on the whole,
- the local consequence finder has been redesigned and optimized in order to improve its performances and to produce only proper prime consequences.

All this characteristics contribute to the increased robustness of this new platform. First experimental results have been reported and seem promising. They suggest that the overhead induced by the multiple layers are easily compensated by the optimization introduced elsewhere.

Our plan is to deepen our experimental study in the spirit of [2] in order to evaluate on which kind of networks and theories these decentralized consequent finder can scale up (or not). Of course, we intend to consider much larger networks of peers than those considered in this paper, but we would also like to consider other kind of structures for local theories. Particularly, since we aim at using SOMEWHERE2 as a building block for PDMS systems, we would like to test these solvers on networks of peers, whose theories are closer to the structure of ontological knowledge bases.

We expect SOMEWHERE2 to be much easier to use, to maintain and to extend, and for the benefit of the community, the code is released under an open source licence Cecill-C (similar to LGPL) through the SourceSup forge [30].

Acknowledgement

We would like to thank the anonymous referees for their comments and suggestions. This work has been supported by INRIA through an IJD grant.

REFERENCES

[1] Abdallah, N.—Goasdoué, F.—Rousset, M.-C.: Dl-Lite$_R$ in the Light of Propositional Logic for Decentralized Data Management. Proceedings of Twenty-First International Joint Conference on Artificial Intelligence (IJCAI), 2009, pp. 2010–2015.

[2] Adjiman, P.—Chatalic, P.—Goasdoué, F.—Rousset, M.-C.—Simon, L.: Distributed Reasoning in a Peer-to-Peer Setting: Application to the Semantic Web. Journal of Artificial Intelligence Research, Vol. 25, January 2006.

[3] Amir, E.—McIlraith, S.: Partition-Based Logical Reasoning. Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR '00), 2000, pp. 389–400.

[4] Amir, E.—McIlraith, S.: Partition-Based Logical Reasoning for First-Order and Propositional Theories. Artificial Intelligence Journal (AIJ), Vol. 162, 2005, No. 1-2, pp. 49–88.

[5] Bell, D. A.—Shao, J.—Hull, M. E. C.: A Pipelined Strategy for Processing Recursive Queries in Parallel. Data & Knowledge Engineering, Vol. 6, 1991, No. 5, pp. 367–391.

[6] Binas, A.—McIlraith, S.: Peer-to-Peer Query Answering with Inconsistent Knowledge. Proceedings of Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR '08), AAAI Press, 2008, pp. 329–339.

[7] Bourgne, G.—Inoue, K.: Partition-Based Consequence Finding. Proceedings of 23rd International Conference on Tools with Artificial Intelligence (ICTAI '11), IEEE, 2011, pp. 641–648.

[8] Calvanese, D.—De Giacomo, G.—Lembo, D.—Lenzerini, M.—Rosati, R.: Tractable Reasoning and Efficient Query Answering in Description Logics: The Dl-Lite Family. Journal of Automated Reasoning, Vol. 39, 2007, No. 3, pp. 385–429.

[9] Chatalic, P.—de Amorim Fonseca, A.: Somewhere2 – A Robust Package for Collaborative Decentralized Consequence-Finding. Proceedings of Intelligent Distributed Computing VII (IDC 2013), Studies in Computational Intelligence, Vol. 511, 2013, pp. 103–108.

[10] Chatalic, P.—Nguyen, G.-H.—Rousset, M.-C.: Reasoning with Inconsistencies in Propositional Peer-to-Peer Inference Systems. Proceedings of European Conference on Artificial Intelligence (ECAI '06), 2006, pp. 352–357.

[11] Gebser, M.—Kaufmann, B.—Neumann, A.—Schaub, T.: Conflict-Driven Answer Set Solving. Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI '07), AAAI Press/The MIT Press, 2007, pp. 386–392.

[12] Grid'5000: A Large Scale and Highly Reconfigurable Grid Experimental Testbed. Web site `https://www.grid5000.fr`.

[13] Halevy, A. Y.—Ives, Z.—Tatarinov, I.—Mork, P.: Piazza: Data Management Infrastructure for Semantic web Applications. Proceedings of the Twelfth International World Wide Web Conference (WWW '03), ACM Press, 2003, pp. 556–567.

[14] Inoue, K.—Bourgne, G.,—Okamoto, T.: Complete Distributed Consequence Finding with Message Passing. Proceedings of the Third International Conference on Agents and Artificial Intelligence (ICAART '11), 2011, pp. 134–143.

[15] Inoue, K.—Bourgne, G.—Okamoto, T.: Distributed Consequence Finding: Partition-Based and Cooperative Approaches Agents and Artificial Intelligence. Communications in Computer and Information Science, Vol. 271, 2013, pp. 429–444.

[16] JGroups: A Toolkit for Reliable Multicast Communication. Web site: `http://www.jgroups.org`.

[17] JXTA$^{\text{TM}}$: The Language and Platform Independent Protocol for P2P Networking. Web site: `http://jxta.kenai.com`.

[18] Kean, A.—Tsiknis, G. K.: An Incremental Method for Generating Prime Implicants/Implicates. Journal of Symbolic Computation, Vol. 9, 1990, No. 2, pp. 185–206.

[19] Kean, A.—Tsiknis, G. K.: A Corrigendum for the Optimized-Ipia. Journal of Symbolic Computation, Vol. 17, 1994, No. 2, pp. 181–187.

[20] Lee, C. T.: A Completeness Theorem and a Computer Program for Finding Theorems Derivable from Given Axioms. Ph.D. thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, USA, 1967.

[21] Marquis, P.: Handbook on Defeasible Reasoning and Uncertainty Management Systems. Vol. 5: Algorithms for Defeasible and Uncertain Reasoning. Chapter Consequence Finding Algorithms. Kluwer Academic Publisher, 2000, pp. 41–145.

[22] Nabeshima, H.—Iwanuma, K.—Inoue, K.—Ray, O.: Solar: An Automated Deduction System for Consequence Finding. AI Communications, IOS Press, Vol. 23, 2010, No. 2-3, pp. 183–203.

[23] Nejdl, W.—Wolf, B.—Qu, C.—Decker, S.—Sintek, M. et al.: EDUTELLA: A P2P Networking Infrastructure Based on RDF. Proceedings of the Eleventh International World Wide Web Conference (WWW '02), ACM Press, 2002, pp. 604–615.

[24] Newman, M. E. J.: Models of the Small World. Journal of Statistical Physics, Vol. 101, 2000, pp. 819–841.

[25] Nguyen, G. H.: Fiabilité des Réponses Fournies par un Réseau Logique Pair-à-Pair. Ph. D. thesis, Université Joseph Fourier, Grenoble, France, 2008.

[26] Wolfson, O.—Dewan, H. M.—Stolofo, S. J.—Yemini, Y.: Incremental Evaluation of Rules and Its Relationship to Parallelism. ACM SIGMOD International Conference on the Management of Data, Denver, Colorado, USA, 1991, pp. 78–87.

[27] Schrag, R.—Crawford, J.: Implicates and Prime Implicates in Random 3-Sat. Artificial Intelligence Journal, Vol. 81, 1996, No. 1-2, pp. 199–222.

[28] Shoal: A Dynamic Clustering Framework. Web site: `http://shoal.java.net`.

[29] Siegel, P.: Représentation et Utilisation de la Connaissance en Calcul Propositionnel. Thèse d'Etat, Université d'Aix-Marseille II, 1987.

[30] Somewhere2 Web site: `https://somewhere2.lri.fr`.

[31] Watts, D. J.—Strogatz, S. H.: Collective Dynamics of 'Small-World' Networks. Nature, Vol. 393, 1998, No. 6684, pp. 440–442.

Philippe Chatalic is Associate Professor at Université Paris-Sud, Orsay, France. He received his M.Sc. and Ph.D. degrees in computer science from Université of Toulouse 3, France (in 1984 and 1986, respectively). His research interests include logical approaches for knowledge representation and reasoning, decentralized reasoning, propositional reasoning, information integration, handling of uncertainty and inconsistency.

Andre de Amorim Fonseca is currently a data mining engineer at Internet Memory. He received his M.Sc. degree in computer science from University of Rennes 1, France in 2010. He has been working as a research and development engineer at Federal University of Bahia, Brazil and at INRIA Saclay, France. His research interests include software engineering, distributed reasoning and data mining.