Efficient Automatic Selection of Semantically-Annotated Building Blocks for ERPs Customizing

Eufemia Tinelli[1,2], Tommaso Di Noia[1], Eugenio Di Sciascio[1], and Francesco di Cugno[1]

[1] SisInfLab, Politecnico di Bari, Via Re David, 200, Bari, Italy
{t.dinoia,disciascio,f.dicugno}@poliba.it
[2] Università di Bari, via Orabona 4, Bari, Italy
tinelli@di.uniba.it

Abstract. We present an approach for efficient semantic-based building-blocks selection in the context of ERPs fast customizing, introducing Enduring Oak, a framework that implements an optimized greedy concept covering algorithm, able to deal with thousands of building block descriptions with reasonable computational times. The proposed approach uses a Description Logics reasoning engine in conjunction with a RDBMS to reduce the computational burden. We motivate the approach, present the framework and algorithms and illustrate experiments confirming the validity of our setting.

1 Introduction

In [7,8] algorithms and a prototype system were proposed for the automated selection of semantically-annotated building blocks in ERPs business processes customization. The framework adopted a polynomial greedy concept-covering algorithm, exploiting non-standard inference services named *concept abduction* and *concept contraction* [9]. The system was specifically designed to ease SAP R/3 best-practices [11] re-usability during the so-called *customizing*. While effective and theoretically sound, the approach —though based on a greedy approach— was unable to efficiently scale-up to the thousands of building blocks descriptions that can be present in a real scenario deployment. In this paper we show that by smoothly combining inference services provided by a reasoning engine with efficient storage and retrieval of relational DBMS we are able to drastically reduce computational times obtaining the desired scalability, with only a limited reduction in the effectiveness of the semantic-based selection process. The remaining of the paper is as follows: next section summarizes the domain we tackle. Then we move on to basics of the formalisms and algorithms we use, in order to make the paper self-contained. Section 4 presents and motivates the approach we propose; In section 5 we briefly outline our semantic-based system and then we report on experiments carried out. Conclusions close the paper.

2 Application Scenario

Enterprise Resource Planning (ERPs) systems have become more and more common in a number of different enterprise and companies; nevertheless they are able to actually

improve the quality and efficiency of companies business processes as long as they are properly tailored and tuned on the actual organization. To this aim they provide a huge number of parametric customizations in order to adapt the system to the various organizational contexts. This process is often much more expensive than the actual purchase of the ERP software [18]. To simplify this stage ERPs producers tend to offer support methodologies and tools to rapidly re-use solutions from previous well established implementations. In particular, for SAP R/3, such process is known as Customizing [2,13]. Customizing methodologies include: Global Accelerated SAP (GASAP), Accelerated SAP (ASAP) and Best Practices [13,11]. Here we focus on the Best Practices methodology, which has —at its core— the Building Block (BB) concept [11]. The basic idea is the modularization of a vertical solution, *i.e.*, in SAP terms a complete SAP R/3 solution developed for a well defined organization scenario, identifying and extracting all its client independent information. BB contents in SAP Best Practices are defined considering from the start the possibility of their reuse from an implementation point of view. Basically, the BB content is defined by the identification of which Business Process parts can be reused within a predefined solution. The BB Library [16] provided by SAP in fact aims at sharing SAP knowledge within the community of developers. It is also possible to develop specific BBs able to provide particular solutions within a company context. Nevertheless, because of the rapid growth of the BBs number, choosing the correct BB in order to satisfy part of a specific Business Process, is increasingly expensive in terms of time, as the selection is driven only by the developers experience. The need to automate such a process, and similar ones, providing support in the customization stage, is therefore increasingly acknowledged. Furthermore we note that, although the system we developed so far —which exploits knowledge representation techniques to automatically select annotated building blocks from available business processes to compose new ones satisfying a given need— has been designed for SAP R/3 BBs, it is obvious that algorithms and solutions devised are applicable to a number of different business processes composition scenarios.

3 Framework and Basic Algorithms

Our semantic-based approach relies on Description Logics (DLs) formalization [3]. Description Logics (DLs) are a family of logic formalisms for Knowledge Representation. We briefly present introductory notions of DLs. Basic syntax elements are: *concept* names, *role* names, *individuals*. Intuitively, concepts stand for sets of objects, and roles link objects in different concepts. Individuals are used for special named elements belonging to concepts. Depending on the expressivity of the language and on the allowed constructors *e.g.*, number restrictions, transitive roles, full negation, etc., different names have been used to identify different Description Logics. Many of them are built upon the simple $\mathcal{AL}$([3]); they all define two special concept names, namely $\top$ and $\bot$ representing respectively, all the objects within the domain and the empty set. Concept expressions can be used in *inclusion assertions*, and *definitions*, which impose restrictions on possible interpretations according to the knowledge elicited for a given domain. Definitions are useful to give a meaningful name to particular combinations. Sets of such inclusions are called TBox (Terminological Box), and amount to what

is usually called an ontology. Individuals can be asserted to belong to a concept using membership assertions in an ABox. The semantics of inclusions and definitions is based on set containment: an interpretation $\mathcal{I}$ satisfies an inclusion $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, and it satisfies a definition $C \equiv D$ when $C^{\mathcal{I}} = D^{\mathcal{I}}$. A *model* of a TBox $\mathcal{T}$ is an interpretation satisfying all inclusions and definitions of $\mathcal{T}$. DL-based systems are equipped with inference services: logical problems whose solution can make explicit knowledge that was implicit in the assertions. Basic inferences are:

Concept Satisfiability — $\mathcal{T} \models C \sqsubseteq \bot$: given an ontology $\mathcal{T}$ and a concept C, does there exist at least one model of $\mathcal{T}$ assigning a non-empty extension to C?

Subsumption — $\mathcal{T} \models C \sqsubseteq D$: given an ontology $\mathcal{T}$ and two concepts C and D, is C more general than D in any model of $\mathcal{T}$?

Given an ontology $\mathcal{T}$ modeling the investigated knowledge domain, a request description D and a resource description BB, in our setting a building block (BB), using subsumption it is possible to evaluate either (1)if BBcompletely fulfills the request – $\mathcal{T} \models BB \sqsubseteq D$, full match – or (2)if they are at least compatible with each other – $\mathcal{T} \not\models BB \sqcap D \sqsubseteq \bot$, potential match – or (3)not – $\mathcal{T} \models BB \sqcap D \sqsubseteq \bot$, partial match [10]. It is easy to see that in case of full match all the information specified on BB is expressed, explicitly or by means of the ontology $\mathcal{T}$, also in D. Responses to calls for subsumption and satisfiability checks are obviously Boolean. When we need to deal with approximation, other inferences may help. Let us consider two concepts C and D;

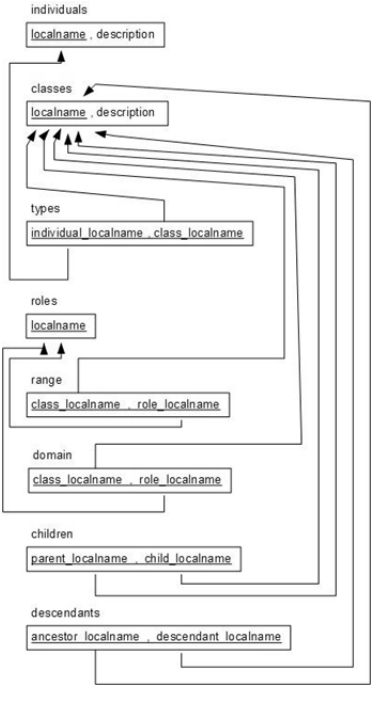

Fig. 1. Relational structure of the database

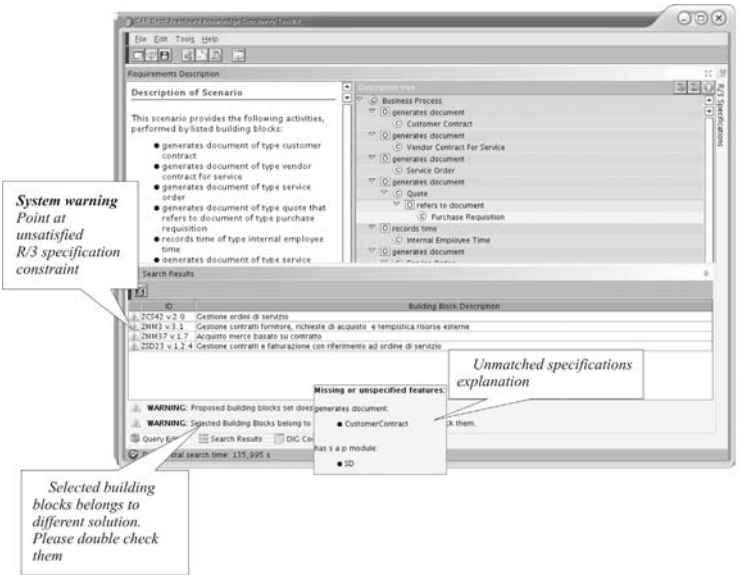

Fig. 2. An annotated snapshot of the User Interface

if their conjunction $C \sqcap D$ is unsatisfiable in the TBox $\mathcal{T}$ representing the ontology, *i.e.*, they are not compatible with each other, we may want, as in a belief revision process, to retract requirements in D, G (for *Give up*), to obtain a concept K (for *Keep*) such that $K \sqcap C$ is satisfiable in $\mathcal{T}$.

Definition 1. *Let $\mathcal{L}$ be a DL, C, D, be two concepts in $\mathcal{L}$, and $\mathcal{T}$ be a set of axioms in $\mathcal{L}$, where both C and D are satisfiable in $\mathcal{T}$. A* Concept Contraction Problem *(CCP), identified by $\langle \mathcal{L}, D, C, \mathcal{T} \rangle$, is finding a pair of concepts $\langle G, K \rangle \in \mathcal{L} \times \mathcal{L}$ such that $\mathcal{T} \models D \equiv G \sqcap K$, and $\mathcal{T} \not\models K \sqcap C \sqsubseteq \bot$. We call K a* contraction *of D according to C and $\mathcal{T}$.*

$\mathcal{Q}$ is a symbol for a CCP, and $SOLCCP(\mathcal{Q})$ denotes the set of all solutions to $\mathcal{Q}$. Obviously, there is always the trivial solution $\langle G, K \rangle = \langle D, \top \rangle$ to whatever CCP, that is give up everything of D. When $C \sqcap D$ is satisfiable in $\mathcal{T}$, the "best" possible solution is $\langle \top, D \rangle$, that is, give up nothing — if possible. When subsumption does not hold *i.e.*, a full match is unavailable, one may want to hypothesize some explanation on which are the causes of this result.

Definition 2. *Let C, D, be two concepts in a Description Logic $\mathcal{L}$, and $\mathcal{T}$ be a set of axioms, where both C and D are satisfiable in $\mathcal{T}$. A* Concept Abduction Problem *(CAP), denoted as $\langle \mathcal{L}, C, D, \mathcal{T} \rangle$, is finding a concept H such that $\mathcal{T} \not\models C \sqcap H \sqsubseteq \bot$, and $\mathcal{T} \models C \sqcap H \sqsubseteq D$.*

$\mathcal{P}$ is a symbol for a CAP, and $SOL(\mathcal{P})$ denotes the set of all solutions to $\mathcal{P}$. Given a CAP $\mathcal{P}$, if H is a conjunction of concepts and no sub-conjunction of concepts in H is a solution to $\mathcal{P}$, then H is an **irreducible solution**.

The *rankPotential* algorithm [10] allows to numerically compute the *length* of H. Recalling the definition, *rankPotential* (C, D) returns to a numerical measure of what is still missing in C w.r.t. D. If $C \equiv \top$ the maximum value for *rankPotential* (C, D) is computed, that is the maximum (potential) mismatch of C from D. Hence, the value returned by *rankPotential*$(\top, D)$ amounts to how specific is a complex concept expression D with respect to an ontology $\mathcal{T}$, what we call the *depth* of D: $\texttt{depth}(D)$.

Concept Covering, originally defined in [12] for a particular set of DLs with limited expressiveness, was later extended and generalized in terms of Concept Abduction in [5]. We recall here this definition:

Definition 3. *Let D be a concept, $\mathcal{R} = \{S_1, .., S_k\}$ be a set of concepts in a Description Logic $\mathcal{L}$ and $\mathcal{T}$ be a set of axioms, where D and $S_i, i = 1..k$ are satisfiable in $\mathcal{T}$.*

1. A Concept Covering Problem *(CCoP), denoted as* CCoP$(\langle \mathcal{L}, \mathcal{R}, D, \mathcal{T} \rangle)$*, is finding, if it exists, a set $\mathcal{R}_c \subseteq \mathcal{R}$, such that both for each $S_j \in \mathcal{R}_c$, $\mathcal{T} \not\models \sqcap S_j \equiv \bot$, and $H \in SOL(\langle \mathcal{L}, \sqcap S_j, D, \mathcal{T} \rangle)$ is such that $H \not\sqsubseteq D$.*
2. We call $\langle \mathcal{R}_c, H \rangle$ a solution *for the CCoP $\langle \mathcal{L}, \mathcal{R}, D, \mathcal{T} \rangle$.*

Intuitively, $\mathcal{R}_c$ is a set of concepts that completely or partially cover D w.r.t. $\mathcal{T}$, while the abduced concept H represents what is still in D and is not covered by $\mathcal{R}_c$. A greedy approach is needed for performance reasons as also the basic set covering problem is NP-Hard. A Concept Covering Problem is similar, but has remarkable differences when compared to classical set covering [7]. There can be several solutions for a single CCoP, depending also on the strategy adopted for choosing candidate concepts in $\mathcal{R}_c$. In [7] the greedy Algorithm 1 was proposed, which we recall here for the sake of clearness. The algorithm *GsCCoP*$(\mathcal{R}, BP, \mathcal{T})$ ha as inputs BP, *i.e.*, the concept expression representing the required Business Process, $\mathcal{R} = \{BB_i\}, i = 1...n$ the set of available Building Blocks descriptions, the reference ontology $\mathcal{T}$.

The algorithm tries to cover BP description "as much as possible", using the concepts $BB_i \in \mathcal{R}$. If a new building block BB_i can be added to the already composed set $\mathcal{R}_c$, *i.e.*, $\mathcal{T} \not\models (\sqcap_{BB_k \in \mathcal{R}_c} BB_k) \sqcap BB_i \sqsubseteq \bot$, then an extended matchmaking process is performed (rows 8–21). If BB_i is not consistent with the uncovered part of the business process BP_u, the latter is contracted and subsequently a CAP is performed between the contracted uncovered part and BB_i (rows 8–10). If BB_i is consistent with BP_u, only a CAP is solved (rows 11–15). Based on the previously computed concepts G, K and H a global score is computed as a metric to evaluate how good BP_i is with respect to the covering set (rows 16–21). The score is determined through Ψ function. It takes into account what has to be given up (G), kept (K) in the uncovered part (BP_u) of BP in order to be compatible BB_i and what has to be hypothesized (H) in order to fully satisfy the contracted BP_u, *i.e.*, K. For $\mathcal{ALN}$DL, that we adopt here the following formula is used [6]:

$$\Psi(G, K, H, BB_i, BP_u) = \left| 1 - \frac{N}{N - g} * (1 - \frac{h}{k}) \right| \tag{1}$$

where N, k, g, h represent numerical evaluation, computed via the $rankPotential$ algorithm[10], for respectively:

Input: $BP, \mathcal{R}$, where BP and all $BB_i \in \mathcal{R}$ are satisfiable in $\mathcal{T}$
Output: $\mathcal{R}_c, \overline{BP}_u, K_{BP}, G_{contraction}$

$BP_u = BP$;
$H_{min} = BP$;
$\mathcal{R}_c = \emptyset$;
repeat
  $BB_{max} = \top$;
  $d_{min} = \infty$;
  foreach $BB_i \in \mathcal{R}$ ***such that*** $\mathcal{R}_c \cup \{BB_i\}$ *covers* BP_u *contraction according to* $\mathcal{Q} = \langle \mathcal{L}, \sqcap_{BB_k \in \mathcal{R}_c} BB_k \sqcap BB_i, BP_u, \mathcal{T} \rangle$ **do**
    if $\mathcal{T} \models BP_u \sqcap BB_i \equiv \bot$ **then**
      $\langle G, K \rangle = contract(BB_i, BP_u, \mathcal{T})$;
      $H = abduce(BB_i, K, \mathcal{T})$;
    else
      $H = abduce(BB_i, BP_u, \mathcal{T})$;
      $K = BP_u$;
      $G = \top$;
    end
    $d = \Psi(G, K, H, BB_i, BP_u)$;
    if $d < d_{min}$ **then**
      $BB_{max} = BB_i$;
      $H_{min} = H$;
      $d_{min} = d$;
    end
  end
  if $(BB_{max} \not\equiv \top)$ **then**
    $\mathcal{R} = \mathcal{R} \backslash \{BB_i\}$;
    $\mathcal{R}_c = \mathcal{R}_c \cup \{BB_i\}$;
    $BP_u = H_{min}$;
  end
until $(BB_{max} \not\equiv \top)$;
$\langle K_{BP}, G_{BP} \rangle = contract(\sqcap_{BB_k \in \mathcal{R}_c} BB_k, BP, \mathcal{T})$;
$\overline{BP}_u = abduce(\sqcap_{BB_k \in \mathcal{R}_c} BB_k, K_{BP}, \mathcal{T})$;
return $(\mathcal{R}_c, \overline{BP}_u, K_{BP}, G_{BP})$;

Algorithm 1. *GsCCoP*$(\mathcal{R}, BP, \mathcal{T})$ – Extended Concept Covering

-BP_u the uncovered part of BP in an intermediate step of the covering process: $N = rankPotential(\top, BP_u, \mathcal{T})$;
-K: belonging to a solution of $\mathcal{Q} = \langle \mathcal{ALN}, BB_i, BP_u, \mathcal{T} \rangle$: $k = rankPotential(\top, K, \mathcal{T})$;
-G: belonging to a solution of $\mathcal{Q} = \langle \mathcal{ALN}, BB_i, BP_u, \mathcal{T} \rangle$: $g = rankPotential(BP_u, K, \mathcal{T})$;
-H: a solution of $\mathcal{P} = \langle \mathcal{ALN}, BB_i, K, \mathcal{T} \rangle$: $h = rankPotential(BB_i, K, \mathcal{T})$.

The outputs of *GsCCoP* are:

-$\mathcal{R}_c$: the set of building blocks selected to compose the business process;
-$\overline{BP}_u$: the part of the business process description that has –in case no exact cover exists– not been covered;
-K_{BP}: the contracted BP;
-G_{BP}: the part of the business process description given-up at the end of the whole composition process.

We note that the proposed approach also allows to explicitly define mandatory requirements M and preferences P within their request as $BP \equiv M \sqcap P$ (as proposed in [6]).

4 Enduring Oak: An Efficient Concept Covering Framework

Several recent approaches try to merge DL-based reasoning with classical Relational DBMS. The objective is obvious: by pre-classifying large knowledge bases on RDBMS it is possible to reduce the burden on inference engines and deal efficiently with large datasets. Such approaches can be classified as either RDF-based, and include [4,17,19,20] or OWL-based [15,14]. In most cases they are limited to the persistent storing of ontologies on RDBMS, and sometimes have limited inference capabilities. We developed a novel Java-based framework, *Enduring Oak*, which was designed specifically to support persistence and the ability to efficiently solve Concept Covering problems. It is compliant with DIG specifications, uses Hypersonic SQL DB for storage and retrieval of persistent data, and calls a DIG-compliant DL reasoner for computing inferences. Currently we adopt MaMaS-tng [1], which implements *rankPotential* and algorithms to solve concept abduction and contraction problems. The database schema (see Figure 1) includes various tables, namely: Individuals, Concepts, Roles, Individual types, Children, Descendants, Roles range, and Roles domain, with the obvious meaning. The relational model we adopted is clearly redundant [1], as *e.g.*, , the *children* table could have been withdrawn by adding a Boolean attribute with a true value assigned whenever a descendant is also a child. Our design choices were guided by the will to improve retrieval efficiency, also at the expenses of normalization, with a greater storage area and a more complex insert mechanism. In the following we illustrate the modified Concept Covering algorithm implemented in the Enduring Oak framework (see Algorithm 2). Before starting to compute the actual Concept Covering, a preprocessing step is needed for the ontology $\mathcal{T}$. For each role R in $\mathcal{T}$ having as range a concept C, add the definition axiom $AUX \equiv \forall R.C$ to $\mathcal{T}$[2]. Notice, that since AUX is a defined concept *i.e.*, it is introduced via an equivalence axiom, the structure of the domain knowledge modeled in the original $\mathcal{T}$ is not modified.

It is easy noticing that the main difference w.r.t. the original algorithm are the calls to the `Candidate Retrieval` procedure (see Algorithm 3). Candidate Retrieval selects a subset of all available BB descriptions in order to avoid checking all the available dataset. Retrieval is carried out combining database queries and inference services calls to the reasoner. Differently from the basic version of the algorithm, using *GsCCoP* a "full" check for covering condition is not needed. In fact, $\mathcal{C}$ contains concepts

[1] http://dee227.poliba.it:8080/MAMAS-tng/

[2] We recall that a range relation can be modeled with the axiom $\top \sqsubseteq \forall R.C$ in $\mathcal{T}$.

Input: $BP, \mathcal{R}$, where BP and all $BB_i \in \mathcal{R}$ are satisfiable in $\mathcal{T}$
Output: $\mathcal{R}_c, \overline{BP}_u, K_{BP}, G_{BP}$

1 $\mathcal{R}_C = \emptyset$;
2 $BP_u = BP$;
3 $H_{min} = BP$;
4 $\mathcal{C} = candidateRetrieval(D, \mathcal{R}, \mathcal{T})$;
5 **repeat**
6 $\quad BB_{MAX} = \top$;
7 $\quad d_{min} = \infty$;
8 $\quad$ **foreach** $BB_i \in \mathcal{C}$ **do**
9 $\quad\quad$ **if** $\mathcal{T} \models BP_u \sqcap BB_i \sqsubseteq \bot$ **then**
10 $\quad\quad\quad \langle G, K \rangle = contract(BB_i, BP_u, \mathcal{T})$;
11 $\quad\quad\quad H = abduce(BB_i, K, \mathcal{T})$;
12 $\quad\quad$ **else**
13 $\quad\quad\quad H = abduce(BB_i, BP_u, \mathcal{T})$;
14 $\quad\quad\quad K = BP_u$;
15 $\quad\quad\quad G = \top$;
16 $\quad\quad$ **end**
17 $\quad\quad d = \Psi(G, K, H, BB_i, BP_u)$;
18 $\quad\quad$ **if** $d < d_{min}$ **then**
19 $\quad\quad\quad BB_{MAX} = BB_i$;
20 $\quad\quad\quad H_{min} = H$;
21 $\quad\quad\quad d_{min} = d$;
22 $\quad\quad$ **end**
23 $\quad$ **end**
24 $\quad$ **if** $(BB_{MAX} \not\equiv \top)$ *and* $(\mathcal{R} \cup \{BB_i\} \cup \mathcal{T})$ *is satisfiable* **then**
25 $\quad\quad \mathcal{R}_c = \mathcal{R}_c \cup \{BB_i\}$;
26 $\quad\quad BP_u = H_{min}$;
27 $\quad\quad \mathcal{C} = candidateRetrieval(BP_u, \mathcal{R}, \mathcal{T})$;
28 $\quad$ **end**
29 **until** $S_{min} \equiv \top)$;
30 $\langle K_{BP}, G_{BP} \rangle = contract(\sqcap_{BB_k \in \mathcal{R}_c} BB_k, BP, \mathcal{T})$;
31 $\overline{BP}_u = abduce(\sqcap_{BB_k \in \mathcal{R}_c} BB_k, K_{BP}, \mathcal{T})$;
32 **return** $(\mathcal{R}_c, \overline{BP}_u, K_{BP}, G_{BP})$;

Algorithm 2. *GsCCoP+*$(\mathcal{R}, D, \mathcal{T})$ – Fast Concept Covering

which are surely consistent with BP_u. What is still to evaluate is if the best concept –from a covering point of view– in $\mathcal{C}$ is consistent with the ones in $\mathcal{R}_c$ (row 24 in Algorithm 2). Then, only a consistency check is needed instead of a full covering check. On the other hand, while computing a covering, the size of $\mathcal{C}$ cannot increase. In fact, $candidateRetrieval$ computes the candidates to be added to $\mathcal{R}_c$ considering only the ones which overlap BP_u. Since BP_u decreases after each iteration, the same is for $\mathcal{C}$. Notice that, due to the preprocessing step, also AUX concepts are taken into account and returned by $candidateRetrieval$, in this way also complex concepts involving roles are taken into account.

Input: D, $\mathcal{R}$, $\mathcal{T}$, where D and all $C_i \in \mathcal{R}$ are satisfiable in $\mathcal{T}$
Output: $\mathcal{C}$

$\mathcal{CN} = \emptyset$;
$\mathcal{A} = parents(D, \mathcal{T})$;
$\mathcal{CN} = \mathcal{A}$;
foreach $CN_j \in \mathcal{A}$ **do**
    $\mathcal{CN} = \mathcal{CN} \cup descendants(CN_j, \mathcal{T})$;
end
foreach $C_i \in \mathcal{R}$ **do**
    foreach $CN \in \mathcal{CN}$ **do**
        if $C_i \sqsubseteq_{\mathcal{T}} CN$ **then**
            $\mathcal{C} = \mathcal{C} \cup \{C_i\}$;
        end
    end
end
return $\mathcal{C}$;

Algorithm 3. $candidateRetrieval(D, \mathcal{R}, \mathcal{T})$ — Candidates Retrieval

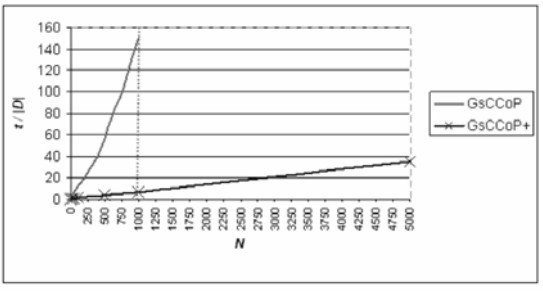

Fig. 3. Comparison of concept covering time performances (GsCCoP: basic *composer* algorithm, GsCCoP+: enhanced *composer* algorithm)

5 System and Performance Evaluation

The architecture of our system comprises MaMaS-tng (which is currently the only reasoner equipped with the above mentioned services), communicating via DIG HTTP and a Hypersonic SQL DBMS interfaced via JDBC. Building Blocks and Business Processes descriptions are all annotated in the $\mathcal{ALN}$ subset of OWL-DL (www.w3.org/TR/owl-features/) language. Obviously, interaction with the system is carried out using a GUI, both for query composition and retrieval of results. The GUI allows to compose a request for a single business process or a composed one. When a fully satisfying result cannot be found, the system returns the degree of completion together with a logical explanation of what remains missing (BP_u) and/or conflicting (G_BP) with the

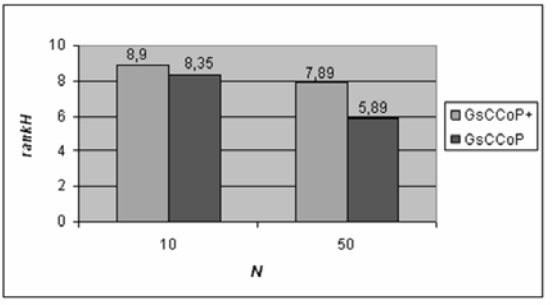

Fig. 4. Comparison w.r.t. the BP_u (uncovered part H)(GsCCoP: basic *composer* algorithm, GsCCoP+: enhanced *composer* algorithm

request, see Figure 2. Extensive experiments aimed at both quantitative and qualitative evaluation have been carried out, comparing the enhanced Enduring Oak approach with the existing system. We note that all experiments presented here have been carried out w.r.t. purely abductive versions of both algorithms, to ease comparison with existing data from previous tests. The test dataset is a randomly generated population of Building Blocks descriptions having a rank with a Gaussian distribution $\mu = 7, \sigma = 5$. Tests have been run using as sample requests descriptions randomly extracted from the dataset, having rank 5, 10, 15, *vs.* sets of 10, 50, 100, 500, 1000, 5000 descriptions randomly extracted from the same dataset at each iteration of the tests. Tests were run on PC platform endowed of a Pentium IV 3.06GHz, RAM 512MB, Java 2 Standard Edition 5.0. We point out that reported results include time elapsed for $\mathcal{T}$ uploading and database connection, with a duration, on average of 5.81 and 0.26 secs, respectively. Obviously such bootstrap times should not be kept into account in the actual usage of the system. Figure 3 presents a comparison between the basic greedy algorithm and the one presented in this paper, clearly showing the computational benefits of the enhanced approach, also in the presence of huge datasets. In both cases, time is normalized w.r.t. to $\texttt{depth}(BP)$. To evaluate the effectiveness of the approach we carried out another set of tests using the same dataset. In this case requests were built as conjunction of a description randomly extracted from the selected dataset with a randomly generated description generated at run time. The rationale is that in this way we ensured that at least a part of the request would have to be satisfied by the description available in the dataset. Results refer to datasets of respectively 10 and 50 descriptions and are summarized in terms of BP_u rank[3], see Figure 4, and covering percentage, see Figure 5, w.r.t. the basic *GsCCoP* and the enhanced *GsCCoP+* algorithms. The tests show that approximations introduced in *GsCCoP+* affect in a limited way the effectiveness of the covering procedure. We note that these are worst case tests, as the population is randomly generated; tests carried out with actual building block descriptions and requests provide results that are practically equivalent for both algorithms.

[3] Recall that BP_u is the part that has to be hypothesized, *i.e.*, what remains uncovered.

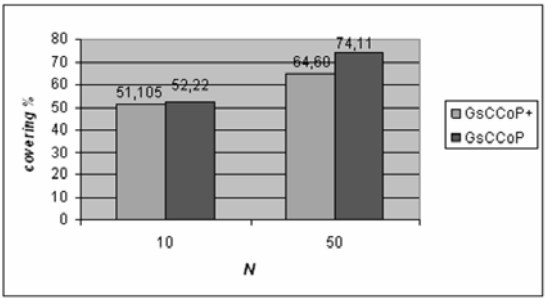

Fig. 5. Successful covering percentage(GsCCoP: basic *composer* algorithm, GsCCoP+: enhanced *composer*+ algorithm

6 Conclusion

While semantic-based systems offer several benefits, they often become inapplicable on real world large datasets. In this paper we have faced such problem in the context of a framework for the automated selection of building-blocks for ERP customizing. Building on a pre-existing solution that implemented a greedy Concept Covering algorithm we have presented an enhanced approach that, partially relying on a relational DBMS, allows to scale our approach up to thousands of BB descriptions with negligible loss in terms of effectiveness.

Acknowledgments

We wish to acknowledge partial support of Apulia regional project ACAB-C2, and EU-FP-6-IST STREP TOWL.

References

1. S Abiteboul, R Hull, and V Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., Reading, Massachussetts, 1995.
2. SAP AG. Contract-based Service Order with Third-Party Services and Time & Material Billing, SAP Best-Practices for Service Providers. help.sap.com/ bestpractices/ industry/serviceindustries/v346c_it/html/index.htm, 2003.
3. F. Baader, D. Calvanese, D. Mc Guinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2002.
4. J. Broekstra, A. Kampman, and F. Harmelen. Sesame: a generic architecture for storing and querying RDF and RDF-Schema. In *proc. of ISWC 2002*, 2002.
5. S. Colucci, T. Di Noia, E. Di Sciascio, F.M. Donini, G. Piscitelli, and S. Coppi. Knowledge Based Approach to Semantic Composition of Teams in an Organization. In *SAC-05*, pages 1314–1319, 2005.
6. S. Colucci, T. Di Noia, E. Di Sciascio, F.M. Donini, and M. Mongiello. Concept Abduction and Contraction for Semantic-based Discovery of Matches and Negotiation Spaces in an E-Marketplace. *Electronic Commerce Research and Applications*, 4(4):345–361, 2005.

7. F. di Cugno, T. Di Noia, E. Di Sciascio, F.M. Donini, and A. Ragone. Concept covering for automated building blocks selection based on business processes semantics. In *Joint 8th IEEE CEC'06 and 3rd EEE'06*, pages 72–79, 2006.
8. F. di Cugno, T. Di Noia, E. Di Sciascio, F.M. Donini, and E. Tinelli. Building-Blocks Composition based on Business Process Semantics for SAP R/3. In *SWCASE'05@ISWC2005*.2005.
9. T. Di Noia, E. Di Sciascio, and F.M. Donini. Semantic Matchmaking as Non-Monotonic Reasoning: A Description Logic Approach. *The Journal of Artificial Intelligence Research*, 2007. To appear.
10. T. Di Noia, E. Di Sciascio, F. M. Donini, and M.Mongiello. A system for Principled Matchmaking in an Electronic Marketplace. *Int. J. of Electronic Commerce*, 8(4):9–37, 2004.
11. R/3 Simplification Group. *Best Practices for mySAP All-in-One, Building Blocks Concept*. 2003.
12. M-S. Hacid, A. Leger, C. Rey, and F. Toumani. Computing Concept Covers: a Preliminary Report. In *DL'02*, volume 53 of *CEUR Workshop Proceedings*, 2002.
13. J. A. Hernández. *SAP R/3 Handbook*. McGraw Hill, 2nd edition, 2000.
14. I Horrocks, L Li, D Turi, and S Bechhofer. The instance store: DL reasoning with large numbers of individuals. In *DL 2004*, pages 31–40, 2004.
15. T. Kessel, M. Schlick, and O. Stern. Accessing configuration-databases by means of description logics. In *KI'95 workshop: knowledge Representation meets Databases*, 1995.
16. SAP Building Block Library. help.sap.com/bestpractices/BBLibrary/bblibrary_start.htm.
17. B. McBride. Jena:implementing the RDF model and syntax specifications. In *SemWeb'01*, 2001.
18. A. Scheer and F Habermann. Making ERP a success. *Comm. of the ACM*, 43(4):57–61, 2000.
19. R. Volz, D Oberle, S Staab, and B Motik. Kaon server - a semantic web management system. In *WWW 2003*, 2003.
20. D. Wood, P. Gearon, and T. Adams. Kowari: a platform for semantic web storage and analysis. In *Proc. of Xtech Conference*, 2005.