Semantic-Based Automated Composition of Distributed Learning Objects for Personalized E-Learning

Simona Colucci[1,3], Tommaso Di Noia[1], Eugenio Di Sciascio[1], Francesco M. Donini[2], and Azzurra Ragone[1]

[1] Politecnico di Bari, Via Re David, 200, I-70125, Bari, Italy
{s.colucci, t.dinoia, disciascio, a.ragone}@poliba.it
[2] Università della Tuscia, via San Carlo, 32, I-01100, Viterbo, Italy
donini@unitus.it
[3] Knowledge Media Institute, The Open University, MK7 6AA, United Kingdom

Abstract. Recent advances in e-learning techonologies and web services make realistic the idea that courseware for personalized e-learning can be built by dynamic composition of distributed learning objects, available as web-services. To be assembled in an automated way, learning objects metadata have to be exploited, associating unambiguous and semantically rich descriptions, to be used for such an automated composition. To this aim, we present a framework and algorithms for semantic-based learning objects composition, fully compliant with Semantic Web technologies. In particular our metadata refer to ontologies built on a subset of OWL-DL, and we show how novel inference services in Description Logics can be used to compose dynamically, in an approximated –but computationally tractable– way learning resources, given a requested courseware description.

1 Introduction

Since the beginnings, the World Wide Web has played a key role in changing the way learning and teaching were delivered. The term e-learning has become common, describing several concepts, from complete web-based courses to distance learning and tutoring.

Recently, also thanks to various standardization efforts [2], emphasis has been placed on the concept of learning object *i.e.*, small and easily reusable educational resources to be composed to allow personalized instruction and courseware creation [21, 9, 4, 30].

Obviously, discovery and composition –according *e.g.*, to prerequisite material– of such learning objects in an automated way requires the association of unambiguous and semantically rich metadata, defined in accordance with shared ontologies. The LOM (Learning Object Metadata) [1] standard, though limited in the basic annotation items, allows to freely define annotated metadata describing a learning resource.

The semantic-based annotation of educational resources is hence fully in the stream of the Semantic Web initiative [29], and it can share with it both techniques and approaches [26, 8, 17]. In particular, as more and more learning objects become available on the Web as services with well-defined machine interpretable interfaces as described

A. Gómez-Pérez and J. Euzenat (Eds.): ESWC 2005, LNCS 3532, pp. 633–648, 2005.

e.g., in OWL-S [28, 22], personalized learning units can be built by scratch, by retrieving learning resources, which are in this scenario semantic-enabled web services themselves [25, 19, 15]. Automated composition of learning resources, exposed as web services for example, can then match a personalized learning need.

In this paper we propose a framework and an approach where, given a specification of courseware described using a subset of OWL-DL [24], we are able to discover and compose, using semantics of descriptions, learning resources covering as much as as possible the learning need, and orchestrating resources according to specified prerequisites. Furthermore, thanks to recently introduced inference services, when available resources are unable to fulfill the needs, our approach provides an explanation of what is missing to fully cover the request. We show how this is obtained by presenting a greedy algorithm for Concept Covering in a subset of OWL-DL and exploit it for semantic service discovery. Our framework also allows to carry out the assembly and integration of learning objects. The greedy algorithm takes into account approximate solutions and is computed in polynomial time.

The remaining of the paper is structured as follows: next section describes basics of Description Logics and the subset of OWL-DL we concentrate on; then we present our extension of Concept Covering [18] definition and a greedy algorithm, which uses Concept Abduction [13], to determine a Concept Cover. In Section 3 we exploit the previously defined Concept Covering algorithm in a general framework to carry out a semantic learning objects assembly and orchestration. The approach, and behavior of the related algorithms, are thoroughly explained with the aid of an example in Section 5. Last Section draws the conclusions and outlines future research directions.

2 Basic of Description Logics

We start with a brief guided tour of Description Logics (DLs)and their interaction with novel languages for the Semantic Web. DLs[5] are a family of logic formalisms whose basic syntax elements are *concept* names, *e.g.*, `WebService`, `ProceduralLanguage`, `Java`, and *role* names, such as `allowedTechnologies`, `hasProgrammingLanguages`. Intuitively, concepts stand for sets of objects, and roles link objects in different concepts. Formally, concepts are interpreted as subsets of a domain of interpretation Δ, and roles as binary relations (subsets of $\Delta \times \Delta$). Basic elements can be combined using *constructors* to form concept and role *expressions*, and each DL has its distinguished set of constructors. Every DL allows one to form a *conjunction* of concepts, usually denoted as $\sqcap$; some DL include also disjunction $\sqcup$ and complement $\neg$ to close concept expressions under boolean operations. Roles can be combined with concepts using *existential role quantification*, *e.g.*, $\texttt{WYSIWYGtool} \sqcap \exists \texttt{allowedTechnologies}.\texttt{WebService}$, which describes a WYSIWYG tool allowing Web services development, and *universal role quantification*, *e.g.*, $\texttt{IDE} \sqcap \forall \texttt{hasProgrammingLanguage}.\texttt{Java}$, which describes a tool to handle only java code. Other constructs may involve counting, as number restrictions: $\texttt{Tool} \sqcap (\leq\ 2\ \texttt{hasProgrammingLanguage}) \sqcap \forall \texttt{hasProgrammingLanguage}.\texttt{OOL}$ expresses a tool allowing at most two different kinds of object oriented languages. Many other constructs can be defined,

increasing the expressive power of the DL, up to n-ary relations [10]. 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. For example, we could impose that a IDE is a WYSIWYG tool using the following inclusion: $\texttt{IDE} \sqsubseteq \exists \texttt{develTool} \sqcap \forall \texttt{develTool}.\texttt{WYSIWYGtool}$. Definitions are useful to give a meaningful name to particular combinations, as in $\texttt{OOP} \equiv \forall \texttt{programming}.\forall \texttt{language}.\texttt{OOL}$. Historically, sets of such inclusions are called TBox (Terminological Box). The basic reasoning problems for concepts in a DL are satisfiability, which accounts for the internal coherency of the description of a concept (no contradictory properties are present), and subsumption, which accounts for the more general/more specific relation among concepts, that forms the basis of a taxonomy. More formally, a concept C is satisfiable if there exists an interpretation in which C is mapped into a nonempty set unsatisfiable otherwise. If a TBox $\mathcal{T}$ is present, satisfiability is relative to the models of $\mathcal{T}$, that is, the interpretation assigning C to a nonempty set must be a model of the inclusions in $\mathcal{T}$. A concept C subsumes a concept D if every interpretation assigns to C a subset of the set assigned to D. Also subsumption is usually established relative to a TBox, a relation that we denote $\mathcal{T} \models C \sqsubseteq D$. Also a TBox can be said satisfiable if there exist at least one model (i.e., an interpretation fulfilling all its inclusions in a nontrivial way).

It is easy to see that $\mathcal{T}$ in DLs represents what is called an ontology in a knowledge representation system. In the rest of the paper we refer to the $\mathcal{ALN}$(**A**ttributive **L**anguage with unqualified **N**umber restrictions) DL, a subset of OWL-DL. Constructs allowed in an $\mathcal{ALN}$ DL are:

- $\top$ *universal concept*. All the objects in the domain.
- $\bot$ *bottom concept*. The empty set.
- A *atomic concepts*. All the objects belonging to the set represented by A.
- $\neg A$ *atomic negation*. All the objects not belonging to the set represented by A.
- $C \sqcap D$ *intersection*. The objects belonging both to C and D.
- $\forall R.C$ *universal restriction*. All the objects participating to the R relation whose range are all the objects belonging to C.
- $\exists R$ *unqualified existential restriction*. There exists at least one object participating in the relation R.
- $(\geq\ n\ R)|(\leq\ n\ R)|(=\ n\ R)$. Respectively the minimum, the maximum and the exact number of objects participating in the relation R.

We use a $simple-TBox$ in order to express the relations among objects in the domain. With a $simple-TBox$ in all the axioms (for both inclusion and definition) the left side is represented by a concept name.

Ontologies using the above logic can be easily modeled using languages for the Semantic Web [12, 23, 24]. These languages have been conceived to allow for representation of machine understandable, unambiguous, description of web content through the creation of domain ontologies, and aim at increasing openness and interoperability in the web environment. The strong relations between DLs and the above introduced languages for the Semantic Web [7] is also evident in the definition of the three OWL sub-languages. OWL-Lite allows class hierarchy and simple constraints on relation between classes; OWL-DL is based on Description Logics theoretical studies, it allows a

Table 1. Correspondence between OWL-DL and $\mathcal{ALN}$ DL syntax

OWL syntax	DL syntax
$< owl : Thing/ >$	$\top$
$< owl : Nothing/ >$	$\bot$
$< owl : Classrdf : ID = "C"/ >$	C
$< owl : ObjectPropertyrdf : ID = "R"/ >$	R
$< rdfs : subClassOf/ >$	$\sqsubseteq$
$< owl : equivalentClass/ >$	$\equiv$
$< owl : disjointWith/ >$	$\neg$
$< owl : intersectionOf/ >$	$\sqcap$
$< owl : allValuesFrom/ >$	$\forall$
$< owl : someValuesFrom/ >$	$\exists$
$< owl : maxCardinality/ >$	$\leq$
$< owl : minCardinality/ >$	$\geq$
$< owl : cardinality/ >$	$=$

great expressiveness keeping computational completeness and decidability; OWL-Full: using such a language, there is a huge syntactic flexibility and expressiveness. This freedom is paid in terms of no computational guarantee.

The subset of OWL-DL Tags allowing to express an $\mathcal{ALN}$ DL is presented in Table 1. In the rest of the paper we will use DL syntax instead of OWL-DL syntax, to make expressions much more compact.

3 Concept Covering in DLs

Standard inference services in DLs include subsumption and satisfiability. These are enough when a yes/no answer is needed. However there are scenarios that require explanation. In [13] the Concept Abduction Problem (CAP) was introduced and defined as a non standard inference problem for DLs to provide an explanation when subsumption does not hold.

Definition 1. *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 $\mathcal{T} \not\models C \sqcap H \equiv \bot$, and $\mathcal{T} \models C \sqcap H \sqsubseteq D$.*

We use $\mathcal{P}$ as a symbol for a CAP, and we denote with $SOL(\mathcal{P})$ the set of all solutions to a CAP $\mathcal{P}$.

In [13] also minimality criteria for H and a polynomial algorithm to find solutions which are irreducible, for an $\mathcal{ALN}$ DL, have been proposed.

Given a CAP, if H is a conjunction of concepts and no sub-conjunction of concepts in H is a solution to the CAP, then H **is an irreducible solution**. The *rankPotential* algorithm [14] allows to numerically compute the *length* of H.

The solution to a CAP can be interpreted as *what has to be hypothesized in C, and in a second step added to, in order to make C more specific than D?* In other words

H is *what is expressed, explicitly or implicitly, in D and is not present in C*, or again *which part of D is not covered by C*. On the basis of the latter remark in the following we will show how to use concept abduction to perform a "concept covering".

In [18] the *best covering problem* in DLs was introduced as "...a new instance of the problem of rewriting concepts using terminologies".

That is, given a concept C and a set of concept definitions in a terminology $\mathcal{T}$, find concepts defined in $\mathcal{T}$ such that their conjunction can be an approximation of C.

In order to define a concept covering two non standard inference services were there used: the least common subsumer (*lcs*)[6] and the *difference* or *subtraction* operation [27]. Unfortunately, as the authors admitted, the difference operator makes sense only for a limited set of DLs, and surely not for the $\mathcal{ALN}$ (we do not delve into details, for a complete description see [27]).

In a more formal way the authors of [18] defined *cover* as follows.

Definition 2. *Let $\mathcal{L}$ be a Description Logic with structural subsumption, $\mathcal{T}$ be a terminology using operator allowed by $\mathcal{L}$, $\mathcal{R}$ be the set of concept definitions in $\mathcal{T}$, $\mathcal{R} = \{S_i, i \in [1..n]\}$, and D be a concept in $\mathcal{L}$ such that $\mathcal{T} \not\models D \equiv \bot$. A* cover *of a D using $\mathcal{T}$ is finding a set $\mathcal{R}_c \subseteq \mathcal{R}$ such that $\sqcap S_i$), conjunction of all the $S_i \in \mathcal{R}_c$ is such that $D - lcs_{\mathcal{T}}(D, \sqcap S_i) \not\equiv D$.*

That is, a cover is finding a set of concepts defined in $\mathcal{T}$ such that they contain the information in D. Notice that a DL with structural subsumption is needed in order to use *concept difference*. In [18] also an hypergraphs based methodology is presented to compute best covers.

We extended the previous definition, in terms of a Concept Covering Problem, both eliminating limitations on $\mathcal{L}$ to be used and rewriting it in terms of Concept Abduction.

Definition 3. *Let D be a concept, $\mathcal{R} = \{S_1, S_2, ..., S_k\}$ 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 $\langle \mathcal{L}, \mathcal{R}, D, \mathcal{T} \rangle$, 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 $\langle \mathcal{L}, \mathcal{R}, D, \mathcal{T} \rangle$.*

In the above definition the elements for the solution $\langle \mathcal{R}_c, H \rangle$ of a CCoP represent respectively:

- $\mathcal{R}_c$. Which concepts in $\mathcal{R}$ represent the cover for D w.r.t. $\mathcal{T}$.
- H. What is still in D and is not covered by concepts in $\mathcal{R}$.

We say that $\mathcal{R}_c$ **covers** D and we use the symbol $\mathcal{V}$ for CCoP and $SOLCCoP(\mathcal{V})$ for the set of all the solution to a CCoP $\mathcal{V}$. Actually, there are several solution for a single CCoP, depending also on the strategy adopted for searching concepts belonging to $\mathcal{R}_c$. Based on the definition of Concept Covering Problem we now define the *best cover* and the *exact cover*.

Definition 4. *Given $\mathcal{V}$, a* best cover *for $\mathcal{V}$, w.r.t. an order $\prec$ for $\mathcal{L}$, is a solution $\langle \mathcal{R}_c, H_b \rangle \in SOLCCoP(\mathcal{V})$ such that there is no other $\langle \mathcal{R}'_c, H' \rangle \in SOLCCoP(\mathcal{V})$ with $H' \prec H_b$.*

There is no solution $\langle \mathcal{R}'_c, H' \rangle$ for $\mathcal{V}$ such that H', the remaining part of D yet to be covered, is *smaller* than H_b.

Definition 5. *Given $\mathcal{V}$, a* full cover *for $\mathcal{V}$ is a solution $\in SOLCCoP(\mathcal{V})$ such that $H_e \equiv \top$.*

Having a set $\mathcal{R}$ of concepts $S_i, i = 1..k$, we want to find a subset $\mathcal{R}_c$ of $\mathcal{R}$, if it exists, such that the conjunction of all the concepts in $\mathcal{R}_c$ is more specific than, *i.e.*, it is subsumed by, D. In other words, we are looking for a set of concepts that completely cover D.

3.1 An Algorithm to Solve a CCoP

It is well known that the general set-covering problem is NP-Hard. Here we adapt a tractable greedy set-covering algorithm [11] to compute a CCoP.

Algorithm $GREEDY\,solveCCoP(\mathcal{R}, D, \mathcal{T})$
input concepts $D, S_i \in \mathcal{R}, i = 1..k$, where D and S_i are satisfiable in $\mathcal{T}$
output $\langle \mathcal{R}_c, H \rangle$
begin algorithm
  $\mathcal{R}_c = \emptyset$;
  $D_{uncovered} = D$;
  $H_{min} = D$;
  do
    $S_{min} = \top$;
    /* [♣] *Perform a greedy search among* $S_i \in \mathcal{R}$ */
    for each $S_i \in \mathcal{R}$
      if $\mathcal{R}_c \cup \{S_i\}$ *is a cover* for $D_{uncovered}$ **then**
        $H = solveCAP(\langle \mathcal{L}, S_i, D_{uncovered}, \mathcal{T} \rangle)$;
        /* [♢] *Choose* S_i *based on an order* */
        if $H \prec H_{min}$ **then**
          $S_{min} = S_i$;
          $H_{min} = H$;
        end if
      end if
    end for each
    /* [♠] *If a new* S_i *is found then add* S_i to $\mathcal{R}_c$
    and remove it from $\mathcal{R}$ */
    if $S_{min} \not\equiv \top$ **then**
      $\mathcal{R} = \mathcal{R} \backslash \{S_i\}$;
      $\mathcal{R}_c = \mathcal{R}_c \cup \{S_i\}$;
      $D_{uncovered} = H_{min}$;
    end if
    /* [♡] *Continue searching until no* S_i *is found* */
  while($S_{min} \not\equiv \top$);
  return $\langle \mathcal{R}_c, D_{uncovered} \rangle$;
end algorithm

The algorithm tries to cover D *as much as possible*, using the concepts $S_i \in \mathcal{R}$.

♡ If no new useful $S_i \in \mathcal{R}$ is found, that is any S_i such that it covers D more, then the algorithm terminates.
♣ A greedy approach is used to choose the *candidates* for $\mathcal{R}_c$.

♢ Choose among the candidates the one such that H, solution for the local CAP, is minimal w.r.t. an order $\prec$.

♠ If the greedy search returns a new S_i, it is removed from $\mathcal{R}$ and added to $\mathcal{R}_c$.

In [11] it is proved that, for a set covering problem, the solution grows logarithmically in the size of the set to be covered with respect to the minimal one. Hence the complexity source is in the solution of the CAPs and the comparison in [♢]. For the $\mathcal{ALN}$ DL, in [13] a polynomial algorithm (*findIrred*) is proposed to find irreducible solutions for a CAP, and in [14] the tractable *rankPotential* is presented to rank concepts. Using such algorithms it can be easily proved that also *GREEDYsolveCCoP* can be solved in polynomial time. Obviously we are not claiming that we solve a covering problem polynomially. The algorithm returns *a cover*, not the best one.

4 Semantic Enabled Learning Object Composition

In the following we present how to exploit the DL standard and non-standard inference services in order to perform an automated composition of Learning Objects to assemble personalized learning objectives. Here we do not refer to a particular model specification but we propose a general framework based on OWL technologies for composition which can be easily integrated in existing metadata specifications, such as SCORM [3], LOM [1], IMS [20], Dublin Core [16].

Hereafter we will refer only to a specific portion of the Learning Object (λ) model, but the approach can be easily extended taking into account all the information in the model.

In our view, the discovery of Learning Objects to be composed is a sub-problem of the more generic resource retrieval one. In this perspective, if there is a *learning request* and a repository of learning objects potentially satisfying the learner specifications, a solution to a λ-retrieval problem is:

retrieve (a sequence of) some λs from the repository such that their composition satisfies the request as far as possible.

Notice that we are not necessarily interested in a full satisfaction of the user request; we want to satisfy it *as much as possible*.

If the system is not able to extract a set of objects from the repository such that they completely fulfill the search goal, an approximate solution has to be taken into account, possibly explaining the approximation.

To introduce and motivate the approach, we start with a model where both the learning request, denoted as ρ, and the λ information needed for discovery are represented by an OWL-DL annotation. Then we will enrich the model adding features to compose the discovered λs.

In the initial model we define both ρ and the description of each learning object λ_D simply as DL concept descriptions w.r.t. an ontology $\mathcal{T}$. We assume the existence of a λs repository, where the all the information related to each λ is stored with respect to a generic learning object model.

Given a λ request ρ modeled w.r.t. an ontology $\mathcal{T}$, the steps needed in order to obtain a set of λs satisfying ρ as much as possible are hence the following:

1. query the repository in order to obtain all the λ descriptions, λ_D, referring to the same $\mathcal{T}$. That is, they could perform the task required by the user.
2. create the set $\mathcal{R}$ collecting all the retrieved λ_D.
3. compute the solution $\langle \mathcal{R}_c, H \rangle$ for the $CCoP$ $\langle \mathcal{L}, \mathcal{R}, \rho, \mathcal{T} \rangle$ using the algorithm $GREEDY\,solveCCoP(\mathcal{R}, \rho, \mathcal{T})$.
4. referring to $\langle \mathcal{R}_c, H \rangle$ computed in the previous step, return to the user both $\mathcal{R}_c = \{\lambda_i\}$, representing the set of learning objects in the repository satisfying ρ, and H as an explanation of what is not specified in any $\lambda_i \in \mathcal{R}_c$.

$\mathcal{R}_c$ and H respectively represent the set of λcorresponding to an approximate solution to the retrieval problem and the explanation why the solution is not an exact one.

Using the above approach a **discovery** process is performed for the λs in the repository, which can be composed in order to satisfy ρ as far as possible. Obviously, the discovery of all the services in the repository is a trivial solution to the problem, of no interest.

The **composition** of the discovered $\lambda_i \in \mathcal{R}_c$ requires further information to be taken into account. Some λ may require background knowledge from the learner. If the user does not hold specific knowledge, then she/he is not able to benefit from the use of λ. The learner can get such information using the previous λ in a composition flow.

In order to benefit from the use of a λ, the user knowledge must satisfy the background knowledge required from λ.

4.1 Background Knowledge for Automated Lesson Composition

In order to deal with the execution information, we extend the previous model and define:

Learning Request: $\rho = \langle \rho_D, \rho_{\mathcal{BK}} \rangle$*, where* ρ_D *is the description of the requested lesson and* $\rho_{\mathcal{BK}}$ *represents the background knowledge owned by the requester before looking for the courseware.*

Learning Object:[1] $\lambda = \langle \lambda_D, \lambda_{\mathcal{BK}} \rangle$. λ_D *describes the knowledge the user will acquire after she/he uses* λ*. Using a language endowed with a well-defined syntax and semantics, it models the* offered knowledge. $\lambda_{\mathcal{BK}}$ *is a representation of prerequisites in order to benefit from* λ*.*

ρ_D, $\rho_{\mathcal{BK}}$, λ_D and $\lambda_{\mathcal{BK}}$ are modeled using OWL DL statements referring to an OWL DL task ontology. Notice that $\rho_{\mathcal{BK}} = \top$ or $\rho_{\mathcal{BK}} = \top$ means, respectively, that the user has not any knowledge related to the field she/he wants to learn about and that no background knowledge is needed in order to benefit from the learning object.

For the sake of clarity, from now on we will model OWL DL expressions using their equivalent DL formulation.

[1] Without loss of generality here we consider only the information needed for a semantic discovery and composition.

A simple covering solution, as the one proposed above, cannot deal with the $\rho_{\mathcal{BK}}$, $\lambda_{\mathcal{BK}}$ specifications of the background knowledge respectively owned by the user and required to benefit from λ. To compose Learning Objects dealing with their required background knowledge, we introduce a definition of composite courseware.
A **courseware flow** with respect to some initial background knowledge $\rho_{\mathcal{BK}}$, denoted as $\overline{\Lambda}(\rho_{\mathcal{BK}})$, is a finite sequence of learning objects $(\lambda^1, \lambda^2, ..., \lambda^i, ..., \lambda^n)$, where for each learning object λ^i belonging to to the courseware, all the following conditions hold:

1. the background knowledge owned by the learner before benefiting from the lesson, $\rho_{\mathcal{BK}}$, is at least $\lambda^1_{\mathcal{BK}}$, that is the background knowledge required by λ^1, the first Learning Object of the sequence. In order to learn from a sequence of learning objects (LOs), the user must have at least the knowledge required to learn from the starting LOs.
2. after using λ^{i-1}, the user has a background knowledge which is at least $\lambda^i_{\mathcal{BK}}$, *i.e.*, the one required by the *i-th* Learning Object. While benefiting of the composite LOs, the user acquires new knowledge which becomes part of her/his background. Such an *updated* background knowledge must satisfy the λ^i requirements.

Now the question is: "What is the background knowledge of the user after she/he benefits from the *(i-1)-th* learning object"?

The background knowledge of the learner before the fruition of λ^i is the conjunction of all the knowledge provided by λ^j_D, with $j < i$, and the initial background knowledge $\rho_{\mathcal{BK}}$.

Indicating with $\mathcal{BK}_i$ the background knowledge before using λ^i, using the DL syntax, the following relation ensues:

$$\mathcal{BK}_i = \rho_{\mathcal{BK}} \sqcap \lambda^1_D \sqcap \lambda^2_D \sqcap ... \sqcap \lambda^{i-1}_D$$

We can now define formally a **courseware flow**.

Definition 6. *A* **courseware flow** *with respect to some initial background knowledge $\rho_{\mathcal{BK}}$ is a finite sequence of learning objects $\overline{\Lambda}(\rho_{\mathcal{BK}}) = (\lambda^1, \lambda^2, ..., \lambda^i, ..., \lambda^n)$ with $i = 1..n$, where for each $\lambda^i \in \overline{\Lambda}(\rho_{\mathcal{BK}})$ all the following conditions hold:*

1. $\rho_{\mathcal{BK}} \sqsubseteq \lambda^1_{\mathcal{BK}}$.
2. $\mathcal{BK}_i \sqsubseteq \lambda^i_{\mathcal{BK}}$.

– We indicate with $\mathcal{D}_{\overline{\Lambda}}$, the set of learning objects descriptions in $\overline{\Lambda}(\rho_{\mathcal{BK}})$. $\mathcal{D}_{\overline{\Lambda}} = \{\lambda^i_D | \lambda^i \in \overline{\Lambda}(\rho_{\mathcal{BK}})\}$.

Based on the previous definition of **courseware flow**, it possible to define a **composite courseware** with respect to a request ρ.

Definition 7. *Let $\mathcal{R} = \{\langle \lambda^i_D, \lambda^i_{\mathcal{BK}} \rangle\}$, with i=1..k, be a set of learning objects λ^i, and $\langle \rho_D, \rho_{\mathcal{BK}} \rangle$ be a request for a courseware, such that λ^i_D, $\lambda^i_{\mathcal{BK}}$, ρ_D and $\rho_{\mathcal{BK}}$ are modeled as concept descriptions in a DL w.r.t. an ontology $\mathcal{T}$.*

A **composite courseware** *for $\rho = \langle \rho_D, \rho_{\mathcal{BK}} \rangle$ with respect to $\mathcal{R}$, denoted $\Lambda(\rho, \mathcal{R})$, is a courseware flow such that for each λ_j in the courseware flow, $\mathcal{D}_\Lambda = \{\lambda^j_D | \lambda^j \in \Lambda(\rho, \mathcal{R})\}$,* covers *$\rho_D$.*

Within a resource retrieval scenario, a composite courseware is a sequence of learning objects such that both the following conditions hold: it can be started using some background knowledge the requester owns ($\rho_{\mathcal{BK}}$) and the provided composite courseware covers the user request description (ρ_D).

4.2 Automated Composite Courseware Generation

We adapt now the previously introduced *GREEDYsolveCCoP* to cope with background knowledge and present an algorithm to automatically compute a composite courseware.

For such purpose we need to define an *usable learning object* and an *usable set*.

Definition 8. *Given a courseware flow $\overline{\Lambda}(\rho_{\mathcal{BK}}) = (\lambda^1, \lambda^2, ..., \lambda^n)$, we say that a learning object is a* **usable learning object** *λ_u for $\overline{\Lambda}(\rho_{\mathcal{BK}})$ if and only if*

1. $\lambda_u \notin \overline{\Lambda}(\rho_{\mathcal{BK}})$.
2. $\overline{\overline{\Lambda}}(\rho_{\mathcal{BK}}) = (\lambda^1, \lambda^2, ..., \lambda^n, \lambda_u)$ *is a courseware flow.*

A **usable learning object** λ_u for $\overline{\Lambda}(\rho_{\mathcal{BK}})$ is a learning object which can be used after the user benefits from $\overline{\Lambda}(\rho_{\mathcal{BK}})$, *i.e.*, its required background knowledge is provided by $\overline{\Lambda}(\rho_{\mathcal{BK}})$.

Actually, given a courseware flow, several usable learning objects exist.

Definition 9. *Given a courseware flow $\overline{\Lambda}(\rho_{\mathcal{BK}})$ and a set of learning objects $\mathcal{R} = \{\lambda^i\}$ we call* **usable set** *for $\overline{\Lambda}(\rho_{\mathcal{BK}})$, the set of all the $\lambda^i \in \mathcal{R}$ such that λ^i is a usable learning object for $\overline{\Lambda}(\rho_{\mathcal{BK}})$}. $\mathcal{U}_{\overline{\Lambda}(\rho_{\mathcal{BK}})} = \{\lambda^i_u | \lambda^i_u$ *is a usable learning object for* $\overline{\Lambda}(\rho_{\mathcal{BK}})\}$

The *usable set* is hence the set of all the learning objects that can be used after the user benefits from a courseware flow.

Algorithm $teacher(\mathcal{R}, \langle \rho_D, \rho_{\mathcal{BK}} \rangle, \mathcal{T})$
input a set of learning objects $\mathcal{R} = \{\lambda^i = \langle \lambda^i_D, \lambda^i_{\mathcal{BK}} \rangle\}$, a request $\rho = \langle \rho_D, \rho_{\mathcal{BK}} \rangle$, where λ^i_D, $\lambda^i_{\mathcal{BK}}$, ρ_D and $\rho_{\mathcal{BK}}$ are satisfiable in $\mathcal{T}$
output $\langle \Lambda, H \rangle$
begin algorithm
  $\Lambda(\rho, \mathcal{R}) = \emptyset$;
  $\rho_{D_{uncovered}} = \rho_D$;
  $H_{min} = \rho_D$;
  do
    compute $\mathcal{U}_{\Lambda(\rho,\mathcal{R})}$;
    $\lambda_{D_{min}} = \top$;
    for each $\lambda^i \in \mathcal{U}_{\Lambda(\rho,\mathcal{R})}$
      if $\mathcal{D}_{\Lambda(\rho,\mathcal{R})} \cup \{\lambda^i_D\}$ *covers* $\rho_{D_{uncovered}}$ **then**
        $H = solveCAP(\langle \mathcal{L}, \lambda^i_D, \rho_{D_{uncovered}}, \mathcal{T} \rangle)$;
        if $H \prec H_{min}$ **then**
          $\lambda_{D_{min}} = \lambda^i_D$;
          $H_{min} = H$;
        end if
      end if
    end for each
    if $\lambda_{D_{min}} \not\equiv \top$ **then**
      $\mathcal{R} = \mathcal{R} \backslash \{\lambda^i\}$;
      $\Lambda(\rho, \mathcal{R}) = (\Lambda(\rho, \mathcal{R}), \lambda^i)$;
      $\rho_{D_{uncovered}} = H_{min}$;
    end if
  while($\rho_{D_{min}} \not\equiv \top$);
  return $\langle \Lambda(\rho, \mathcal{R}), \rho_{D_{uncovered}} \rangle$;
end algorithm

The algorithm returns the composite courseware $\Lambda(\rho, \mathcal{R})$ and the uncovered part, $\rho_{D_{uncovered}}$, of the request description ρ_D. The main difference between $GREEDY solveCCoP$ and $teacher$ is that the learning objects to be added to the lesson flow are searched for only within the current *usable learning objects*.

5 Example

In this section we show, with the aid of an example, the behavior of *teacher* with respect to a scenario related to Computer Science teaching. In the example we will refer to the toy ontology in Figure 1 in order to model ρ_D, $\rho_{\mathcal{BK}}$, λ_D and $\lambda_{\mathcal{BK}}$. We refer to a

$\texttt{WYSIWYGtool} \sqsubseteq \texttt{Tool}$
$\texttt{IDE} \sqsubseteq \exists \texttt{develTool} \sqcap \forall \texttt{develTool}.\texttt{WYSIWYGtool}$
$\texttt{OOL} \sqsubseteq \texttt{ProgrammingLanguage}$
$\texttt{ProceduralLanguage} \sqsubseteq \texttt{ProgrammingLanguage}$
$\texttt{OOL} \sqsubseteq \neg \texttt{ProceduralLanguage}$
$\texttt{Java} \sqsubseteq \texttt{OOL}$
$\texttt{C++} \sqsubseteq \texttt{OOL}$
$\texttt{Java} \sqsubseteq \neg \texttt{C++}$
$\texttt{OOP} \equiv \exists \texttt{language} \sqcap \forall \texttt{language}.\texttt{OOL}$
$\texttt{WebService} \sqsubseteq \texttt{DistributedTechnology}$
$\texttt{CORBA} \sqsubseteq \texttt{DistributedTechnology}$
$\texttt{WebService} \sqsubseteq \neg \texttt{CORBA}$

Fig. 1. The ontology used as reference in the example

student wishing to learn about Java programming language with the aim of web service developing on a Unix platform, if specified. She/he has not any knowledge in that field.

$\rho_D = \exists \texttt{language} \sqcap \forall \texttt{language}.\texttt{Java} \sqcap \exists \texttt{allowedTechnologies} \sqcap$
$\forall \texttt{allowedTechnologies}.\texttt{WebService} \sqcap \forall \texttt{OS}.\texttt{Unix}$
$\rho_{\mathcal{BK}} = \top$

The repository contains six learning objects related to the Computer Science domain. $\mathcal{R} = \{\lambda^a, \lambda^b, \lambda^c, \lambda^d, \lambda^e, \lambda^f\}$.

λ^a relates to Object Oriented Languages. It is addressed to beginners.
$\lambda^a_D = \exists \texttt{language} \sqcap \forall \texttt{language}.\texttt{OOL}$
$\lambda^a_{\mathcal{BK}} = \top$

λ^b relates to Java language programming with the aid of an Integrated Development Tool. It is addressed to Object Oriented programmers.
$\lambda^b_D = \exists \texttt{language} \sqcap \forall \texttt{language}.\texttt{Java} \sqcap \texttt{IDE}$
$\lambda^b_{\mathcal{BK}} = \texttt{OOP}$

λ^c relates to C++ Languages programming. It is addressed to Object Oriented programmers.

$\lambda^c_D = \exists \texttt{language} \sqcap \forall \texttt{language.C++}$
$\lambda^c_{\mathcal{BK}} = \texttt{OOP}$

λ^d relates to the CORBA technology. It is addressed to Object Oriented programmers.
$\lambda^d_D = \exists \texttt{allowedTechnologies} \sqcap \forall \texttt{allowedTechnologies.CORBA}$
$\lambda^d_{\mathcal{BK}} = \texttt{OOP}$

λ^e relates to Web Services development with the aid of a WYSIWYG tool. It is addressed to Java programmers.
$\lambda^e_D = \exists \texttt{develTool} \sqcap \forall \texttt{develTool.WYSIWYGtool} \sqcap \exists \texttt{allowedTechnologies} \sqcap \forall \texttt{allowedTechnologies.WebService}$
$\lambda^e_{\mathcal{BK}} = \exists \texttt{language} \sqcap \forall \texttt{language.Java}$

λ^f is an introduction to distributed programming for beginners.
$\lambda^f_D = \exists \texttt{allowedTechnologies} \sqcap \forall \texttt{allowedTechnologies.DistributedTechnology}$
$\lambda^f_{\mathcal{BK}} = \top$

The first step in order to compute $\Lambda(\rho, \mathcal{R})$ is to identify the initial $\mathcal{U}_{\Lambda(\rho,\mathcal{R})}$ with respect to an empty courseware flow $\Lambda(\rho, \mathcal{R}) = \emptyset$ ($\mathcal{U}_0$ for short). As $\rho_{\mathcal{BK}} = \top$ then

$$\mathcal{U}_0 = \{\lambda^a, \lambda^f\}$$

At this initial step $\rho_{D_{uncovered}} = \rho_D$ then:

$H_{\lambda^a} = \exists \texttt{allowedTechnologies} \sqcap \forall \texttt{allowedTechnologies.WebService} \sqcap \forall \texttt{language.Java} \sqcap \forall \texttt{OS.Unix}$
$|H_{\lambda^a}| = 5$

$H_{\lambda^f} = \exists \texttt{language} \sqcap \forall \texttt{allowedTechnologies.WebService} \sqcap \forall \texttt{language.Java} \sqcap \forall \texttt{OS.Unix}$
$|H_{\lambda^f}| = 6$

$\Lambda(\rho, \mathcal{R})$, $\rho_{D_{uncovered}}$ and $\mathcal{BK}_1$ are now respectively:

- $\Lambda(\rho, \mathcal{R}) = (\lambda^a)$
- $\rho_{D_{uncovered}} = \exists \texttt{allowedTechnologies} \sqcap \forall \texttt{allowedTechnologies.WebService} \sqcap \forall \texttt{language.Java} \sqcap \forall \texttt{OS.Unix}$
- $\mathcal{BK}_1 = \exists \texttt{language} \sqcap \forall \texttt{language.OOL}$

With respect to $\mathcal{BK}_1$, the new usable set $\mathcal{U}_1$ is:

$$\mathcal{U}_1 = \{\lambda^b, \lambda^c, \lambda^d, \lambda^f\}$$

With respect to $\mathcal{U}_1$ we have the following values:

$H_{\lambda^b} = \exists \texttt{allowedTechnologies} \sqcap \forall \texttt{allowedTechnologies.WebService} \sqcap \forall \texttt{OS.Unix}$
$|H_{\lambda^b}| = 4$

H_{λ^c} =NOT COMPUTED
$|H_{\lambda^c}|$ =NOT COMPUTED

H_{λ^d} =NOT COMPUTED
$|H_{\lambda^d}|$ =NOT COMPUTED

H_{λ^f} = $\forall$allowedTechnologies.WebService $\sqcap$ $\forall$language.Java $\sqcap$ $\forall$OS.Unix
$|H_{\lambda^f}| = 5$

Both H_{λ^c} and H_{λ^d} are not computed because the information provided by λ^c and λ^d is not compatible with the uncovered one and this situation is not allowed by the Concept Abduction definition. Respectively, λ^c offers knowledge on C++ and the user is looking for Java while λ^d is about CORBA while the user is looking for WebService. The updated $\Lambda(\rho, \mathcal{R})$, $\rho_{D_{uncovered}}$ and $\mathcal{BK}_2$ are

- $\Lambda(\rho, \mathcal{R}) = (\lambda^a, \lambda^b)$
- $\rho_{D_{uncovered}}$ = $\exists$allowedTechnologies $\sqcap$ $\forall$allowedTechnologies.WebService $\sqcap$ $\forall$OS.Unix
- $\mathcal{BK}_2 = \exists$language $\sqcap$ $\forall$language.Java $\sqcap$ IDE

For the next step the new usable set is:

$$\mathcal{U}_2 = \{\lambda^c, \lambda^d, \lambda^e, \lambda^f\}$$

Now, with respect to $\mathcal{U}_2$ we have :

H_{λ^c} =NOT COMPUTED
$|H_{\lambda^c}|$ =NOT COMPUTED

H_{λ^d} =NOT COMPUTED
$|H_{\lambda^d}|$ =NOT COMPUTED

$H_{\lambda^e} = \forall$OS.Unix
$|H_{\lambda^e}| = 1$

$H_{\lambda^f} = \forall$allowedTechnologies.WebService $\sqcap$ $\forall$OS.Unix
$|H_{\lambda^f}| = 2$

Again H_{λ^c} and H_{λ^d} are not computed, but this time the reason is related to the definition of Concept Covering. In fact the description of both λ^c and λ^d is not compatible with the one belonging to the conjunction of λ^a and λ^b.

- $\Lambda(\rho, \mathcal{R}) = (\lambda^a, \lambda^b, \lambda^e)$
- $\rho_{D_{uncovered}} = \forall$OS.Unix
- $\mathcal{BK}_2$ = $\exists$language $\sqcap$ $\forall$language.Java $\sqcap$ IDE $\sqcap$ $\exists$allowedTechnologies $\sqcap$ $\forall$allowedTechnologies.WebService

It is easy to show that, at this point $teacher(\mathcal{R}, \langle \rho_D, \rho_{\mathcal{BK}} \rangle, \mathcal{T})$ stops and it returns:

$$\langle \Lambda, H \rangle = \langle (\lambda^a, \lambda^b, \lambda^e), \forall \texttt{OS.Unix} \rangle$$

The proposed courseware, with respect to the available λ^i in the repository, for the request ρ is then $(\lambda^a, \lambda^b, \lambda^e)$, but ρ is not completely satisfied by the returned Λ because nothing is specified about the platform that will be used ($\forall \texttt{OS.Unix}$).

A prototype system integrated in the MAMAS framework (http://193.204.59.227:8080/MAMAS-devel) has been developed implementing $teacher$ algorithm and exploiting standard (Semantic) Web technologies. The message exchanging is performed using SOAP messages. The semantic information in the body of such messages is formatted using OWL as explained is Section 2.

6 Conclusion and Future Work

In this work we proposed a tractable greedy algorithm to perform an automated courseware composition compliant with the standard Semantic Web technologies and exploiting standard and novel non-standard inference services for Description Logics. We presented motivations and examples for the approach.

We showed how a semantic specification, formatted in OWL-DL, of the learning objects (LOs) can be used both to retrieve from a repository LOs satisfying a user request and to compose such discovered LOs in a courseware.

The proposed approach also copes with non-exact solutions to the courseware composition. That is, if it is not possible to compose, using available LOs, a courseware which completely satisfies the user request, approximate solutions are proposed endowed with an explanation for the non-exact match.

Currently we are developing ontologies related to different tasks, in order to perform further experiments on real scenarios. Under development is also the integration of the implemented prototype with the SCORM specifications.

References

1. *IEEE Standard for Learning Object Metadata*, std 1484.12.1-2002 edition, 2002.
2. *IEEE standard for learning technology-learning technology systems architecture (LTSA)*, std 1484.1-2003 edition, 2003.
3. Advanced Distributed Systems (ADL) Lab, Sharable Content Object Reference Model (SCORM). http://www.adlnet.org/index.cfm?fuseaction=scormabt.
4. K. Ajami. Specifying and implementing interoperable and reusable learning objects: one step beyond. In *Proc. of Intl. Conf. on Information and Communication Technologies: From Theory to Applications*, pages 111–112, 2004.
5. F. Baader, D. Calvanese, D. Mc Guinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2002.
6. Franz Baader. Least common subsumers and most specific concepts in a description logic with existential restrictions and terminological cycles. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI2003)*, pages 319–324, 2003.

7. Franz Baader, Ian Horrocks, and Ulrike Sattler. Description logics as ontology languages for the semantic web. In Dieter Hutter and Werner Stephan, editors, *Festschrift in honor of Jörg Siekmann*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2003.
8. N. Bennacer, Y. Bourda, and B. Doan. Formalizing for querying learning objects using OWL. In *Proc. of Intl. Conf. on Advanced Learning Technologies*, pages 321–325. IEEE, 2004.
9. A.S. Cabezuelo and J.M.D Beardo. Towards a model of quality for learning objects. In *Proc. of Intl. Conf. on Advanced Learning Technologies*, pages 822–825. IEEE, 2004.
10. D. Calvanese, G. De Giacomo, and M. Lenzerini. On the Decidability of Query Containment under Constraints. In *Proceedings of the Seventeenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.
11. Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The Massachusetts Institute of Technology, 1990.
12. DAML+OIL Specifications. www.daml.org/2001/03/daml+oil-index.html, 2001.
13. T. Di Noia, E. Di Sciascio, F.M. Donini, and M. Mongiello. Abductive matchmaking using description logics. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI 2003)*, pages 337–342, Acapulco, Messico, August 9–15 2003. Morgan Kaufmann, Los Altos.
14. T. Di Noia, E. Di Sciascio, F.M. Donini, and M. Mongiello. A system for principled Matchmaking in an electronic marketplace. In *Proc. International World Wide Web Conference (WWW '03)*, pages 321–330, Budapest, Hungary, May 20–24 2003. ACM, New York.
15. Peter Dolog, Nicola Henze, Wolfgang Nejdl, and Michael Sintek. Student tracking and personalization: Personalization in distributed e-learning environments. In *Proc. International World Wide Web Conference (WWW '04)*, 2004. Alternate track papers and posters.
16. Dublin Core Metadata Element Set, Version 1.1: Reference Description . http://dublincore.org/documents/1999/07/02/dces/.
17. D. Gasevic, J. Jovanovic, and V. Devedzic. Enhancing learning object content on the semantic web. In *Proc. of Intl. Conf. on Advanced Learning Technologies*, pages 714–716. IEEE, 2004.
18. Mohand-Sad Hacid, Alain Leger, Christophe Rey, and Farouk Toumani. Computing Concept Covers: a Preliminary Report. In *Proc. of the 15th Intl. Workshop on Description Logics (DL'02)*, volume 53 of *CEUR Workshop Proceedings*, 2002.
19. Marek Hatala, Griff Richards, Timmy Eap, and Jordan Willms. Sharing educational resources: The interoperability of learning object repositories and services: standards, implementations and lessons learned. In *Proc. International World Wide Web Conference (WWW '04)*, 2004. Alternate track papers and posters.
20. IMS, Learning Resource Meta-data Best Practices and Implementation Guide Version 1.1 - Final Specification . http://www.imsproject.org/metadata/mdbestv1p1.html.
21. A. Ip, A. Young, and I. Morrison. Learning objects - Whose are they? In *Proc. of 15 th Conf. of the National Advisory Committee on Computing Qualifications*, pages 315–320, 2002.
22. Sycara Katia, Paolucci Massimo, Ankolekar Anupriya, and Naveen Srinivasan. Automated Discovery, Interaction and Composition of Semantic Web Services. *Journal of Web Semantics*, 1, December 2003.
23. D.L. McGuinness, R. Fikes, J. Hendler, and L.A. Stein. DAML+OIL: An Ontology Language for the Semantic Web . *IEEE Intelligent Systems*, 17(5):72–80, 2002.
24. OWL. www.w3.org/TR/owl-features/.

25. C. Pahl and R. Barrett. A web services architecture for learning object discovery and assembly. In *Proc. International World Wide Web Conference (WWW '04)*, 2004. Alternate track papers and posters.
26. S. Sanchez and M. Sicilia. On the semantics of aggregation and generalization in learning object contracts. In *Proc. of Intl. Conf. on Advanced Learning Technologies*, pages 425–429. IEEE, 2004.
27. G. Teege. Making the difference: A subtraction operation for description logics. In *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR'94)*, pages 540–550. MK, 1994.
28. The OWL Services Coalition . www.daml.org/services/owl-s/1.0/owl-s.html, 2004.
29. James Hendler Tim Berners-Lee and Ora Lassila. The semantic web. *Scientific American*, 248(4), 2001. (34-43).
30. G. Vossen and P. Jaeschke. Learning objects as a uniform foundation for e-learning platforms. In *Proc. of Intl. Symp. on Database Engineering and Applications Symposium*, pages 278–287, 2003.