FONTE - Factorizing ONTology Engineering complexity

Jorge Santos
GECAD - Knowledge Engineering and
Decision Support Research Group
Instituto Superior de Engenharia do Porto
Departamento de Engenharia Informática
4200-072 Porto - Portugal
http://www.dei.isep.ipp.pt/˜jsantos

Steffen Staab
Universität Karlsruhe (TH)
Institut AIFB
D-76128 Karlsruhe - Germany
http://www.aifb.uni-karlsruhe.de/˜sst

ABSTRACT

Because it is difficult to engineer a complex ontology with time, we here consider a method that allows for factorizing the complexity of the engineering process, FONTE (Factorizing ONTology Engineering complexity). FONTE divides the engineering task into building a time-less domain ontology and a temporal theory independently from each other. FONTE provides an operator $\otimes$ that assembles the two independently developed ontologies into the targeted ontology. We investigate the quality of the proposed operator $\otimes$ by applying it to a practical case study, viz. the engineering of an ontology about researchers including temporal interactions.
Categories and Subject Descriptors: I.2.4 Knowledge Representation Formalisms and Methods

1. INTRODUCTION

In recent years, we have seen a surge of ontologies and ontology technology with many ontologies now being available on the Web. At the same time one could observe that most ontologies (e.g., consider the DAML ontology library at `http://www.daml.org/ontologies/`) engineered exhibit
only rather simple structures, *viz.* taxonomies and frame-like links between concepts.

This observation might indicate that such — comparatively — simple structures are sufficient for the large majority of ontology-based systems. According to our own experiences about ontologies for knowledge portals [22] and power systems [18], however, one frequently needs intricate concept descriptions and interactions — in particular ones about time and space.

Because of intense (and fruitfully ongoing) research, many practical theories about time are now well understood. While the same can be said about the engineering of concept hierarchies and concept frames, the issue of how to engineer complex ontologies with intricate interactions based on time has not been researched very deeply, yet, rendering the engineering of a new complex domain ontology with time a labor intensive, one-off experience with little methodology.

K-CAP'03, October 23–25, 2003, Sanibel Island,Florida, USA.

In this paper, we present our ontology engineering methodology, FONTE (Factorizing ONTology Engineering complexity), that pursues a 'divide-and-conquer' strategy for engineering complex ontologies with time. FONTE divides a targeted ontology that is complex and includes time into two building blocks, a temporal theory and a time-less domain ontology. Each one of the two subontologies can be built independently allowing for a factorization of complexity. The targeted ontology is then assembled from the time-less domain ontology and the temporal theory by the operator $\otimes$.

Thereby, the assembling operator $\otimes$ is very different from existing operators for merging or aligning ontologies [15, 17]. Merging ontologies is a process that intends to join different ontologies about overlapping domains into a new one and most of its problems and techniques are related to the identification of similar concepts through structure analysis (e.g. graph analysis, path length, common nodes or/and edges and lexical analysis). For instance, car from ontology 1, *O1.car*, and auto from ontology 2 *O2.auto* may be defined to be identical in the merging process because of results of the structure analysis. To formalize the merging and aligning process, Wiederhold proposed a general algebra for composing large applications through merging ontologies of related domains [26] and actually, the operations proposed (*Intersection*, *Union* and *Difference*) are about the similarities and differences of two ontologies.

In contrast, the result of $\otimes$ needs rather to be seen in rough analogy to the Cartesian product of two entities. For instance, car from ontology 1, *O1.car*, with its frame *O1.licensedInState* is assembled by $\otimes$ with ontology 2 and its *O2.timeInterval* in a way such that every car in the result ontology has a lifetime as well as multiple *O1.licensedInState*-frames with different, mutually exclusive life spans.

⊗ is a non-deterministic operator. It does not have a closed definition[1], but it is operationalized by an iterative, interactive process. It starts off with a human assembly — in the sense just explained — between an ontology *O1*, the time-less domain ontology, and an ontology *O2*, the temporal theory. It is then propelled by a set of rules and a set of constraints. The set of rules drives a semi-automatic process proposing combinations. The set of constraints narrows down the set of plausible proposals to valid ones.

We have applied the methodology FONTE with its operator ⊗ to two case studies. In case study A *O1* is a time-less ontology about a semantic web research community and *O2* is a temporal ontology. In case study B we have used the same temporal theory, but a time-less ontology about the soccer domain. Both times we have investigated how many assembling steps were proposed and we have evaluated their adequacy. The study results suggest that indeed FONTE provides a way to factorize the complexity of building large ontologies with time leading to more reliable and cheaper final products.

The rest of the paper is organized as follows. We first sketch the temporal ontology and the time-less domain ontologies we have used for our case studies. Then we describe the semi-automatic process of assembling two ontologies by ⊗, with some emphasis on the tool support developed to drive the process. Then we give an evaluation of our sample cases, before we relate to other work and conclude with how our approach fits into the larger objectives of having an ontology algebra as proposed by Wiederhold [26].

2. TEMPORAL ONTOLOGY

The temporal ontology used in our case study (see Fig.1 for the UML-like depiction of an excerpt) embodies many concepts like *Instant* or *Period* often found in 'standard' ontologies like Time-DAML [9] or SUMO [14] and assumes a standard interpretation by interpreting time points and time intervals as real numbers and intervals on the real line.

As argued by [24] a temporal representation requires the characterization of time itself and temporal incidence, which are represented in our temporal ontology by *TemporalEntity* and *Eventuality*, respectively. We defined a further notion, *TimedThing*, that bridges between temporal concepts and the domain concepts that will be used during the assemble process. In particular, we have included the notion of *Role* as a core concept. While there are concepts that give identity to their instances (i.e. they are *semantically rigid*), e.g. while the identity of a particular person depends on being an instance of *Person*, the identity of the same person does not change when it ends being a *student* and starts being a *professor*. Thus, the notion of *Role* is important when connecting a temporal theory, e.g. Allen's interval calculus, with a concrete domain, e.g. an ontology about researchers (cf., e.g., [11, 20, 7, 23]).

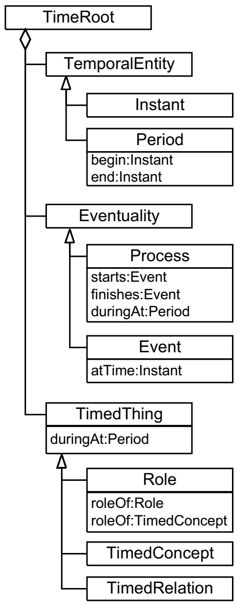


Figure 1: Excerpt of Temporal Ontology Depicted in UML Style

Some of the choices for the temporal ontology that we made were driven by modelling objectives particular to the knowledge portal application that we had in mind when performing the case study. For instance, our objective was to have a 3-dimensional model of the world with time as an extra variable. For future applications an ontology engineer may prefer a 4-dimensional view, leading to a somewhat different temporal ontology and, thus, to an overall different target ontology in the end (cf. [8]).

In our case studies we have not directly exploited Time-DAML or SUMO because both lack the explicit notion of *role*. We consider this notion crucial for ontologies embodying a common three dimensional world model with time [23]. Also it allows us to show some complex re-structuring initiated by ⊗ that we deem of high interest to common applications like knowledge portals.

Nevertheless, this paper shows a factorization of complexity into two subontologies and a re-assembly into a target ontology in a way that is independent of such concrete assumptions. Hence, we may conjecture with high plausibility that our experimental results presented later are also applicable to other theories, such as the slightly different Time-DAML theory or a tremendously different 4-dimensional conceptualization of the world.

Temporal Entities In the temporal ontology we used for the case study there are two subclasses of *TemporalEntity*: *Instant* and *Period*. The relations *before*, *after* and *equality* can hold between *Instants*, respectively represented by the

[1]We discuss the (dis-)advantages of this formulation of ⊗ in the conclusion.

symbols: $\prec$, $\succ$, $=$, allowing to define an algebra based on points [25]. It is assumed that the *before* and *after* are strict linear, namely irreflexive, asymmetric, transitive and linear. The thirteen binary relations proposed in the Allen's interval algebra [1] can be defined in a straightforward way based on the previous three relations [6]. *begin* and *end* are relations from *TemporalEntity* to *Instant*. Also, there are no null duration periods and each period is unique.

Processes and Events. There are two subclasses of *Eventuality*, *Process* and *Event*, in order to be possible to express continuous and instantaneous eventualities, respectively. *Event* has a relation *atTime* to *Instant* while *Process* has a relation *duringAt* to *Period*. The relations *starts* and *finishes* allows to state what can start or finish a process.

Roles. A role can have roles but can not be role of itself (R1) and the *roleOf* relation is transitive (R2). Guarino distinguishes roles from natural types based on the lack of semantic rigidity of the first. The lifespan of a role is timewise contained by the lifespan of the concept or role from which it is a subrole and from which it possibly inherits identity (R3).

$$\forall x : \perp \leftarrow \mathit{roleOf}(x, x). \tag{R1}$$

$$\begin{aligned}\forall x, y, z : \mathit{roleOf}(x, z) \leftarrow \\ \mathit{roleOf}(x, y) \wedge \mathit{roleOf}(y, z).\end{aligned} \tag{R2}$$

$$\begin{aligned}\forall o, x, y, p1, p2 : \mathit{containedBy}(p1, p2) \leftarrow \\ (\mathit{isa}(y, \mathit{Role}) \vee \mathit{isa}(y, \mathit{TimedConcept})) \wedge \\ \mathit{isa}(x, \mathit{Role}) \wedge \mathit{roleOf}(x, y) \wedge \\ \mathit{hasRoleDuringAt}(o, x, p1) \wedge \\ \mathit{hasRoleDuringAt}(o, y, p2).\end{aligned} \tag{R3}$$

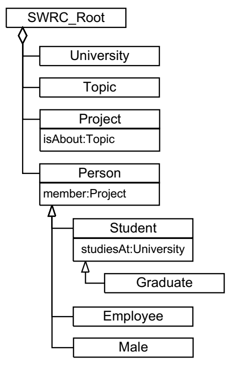
3. CASE STUDY A — THE SWRC ONTOLOGY

The assemble process may be used either for development of ontologies with time from scratch as well as for re-engineering existing ones in order to include time. For our case study we have used the time-less SWRC (Semantic Web Research Community) ontology (`http://ontobroker.semanticweb.org/ontos/swrc.html`) that served as a seed ontology for the knowledge portal of OntoWeb.

SWRC comprises 55 concepts, 151 relations and 25 axioms. The results presented in the section 6 are based on re-engineering the complete SWRC. Here we present an excerpt that is also used in order to elucidate the assembling process with $\otimes$.

4. CASE STUDY B — THE SOCCER ONTOLOGY

This soccer ontology mostly describes concepts that are specific to soccer, like players, rules, field, supporters, actions. It is used to annotate videos in order to produce personalized summary of soccer matches (`http://www.lgi2p.ema.fr/~ranwezs/ontologies/soccerV2.0.daml`).



$\mathit{isa}(\mathit{Student}, \mathit{Person})$
$\mathit{isa}(\mathit{Employee}, \mathit{Person})$
$\mathit{isa}(\mathit{Male}, \mathit{Person})$
$\mathit{isa}(\mathit{Graduate}, \mathit{Student})$
$\mathit{studiesAt}(\mathit{Student}, \mathit{University})$
$\mathit{member}(\mathit{Person}, \mathit{Project})$
$\mathit{isAbout}(\mathit{Project}, \mathit{Topic})$

$$\forall p, t : \mathit{isAbout}(p, t) \leftrightarrow \mathit{dealsWithIn}(t, p) \tag{I1.1}$$

(I1.1 means "*isAbout* is inverse to *dealsWithIn*)

$$\begin{aligned}
&\forall pers, top : \mathit{worksOn}(pers, top) \leftarrow && \text{(A1.1)}\\
&\quad \exists proj : \mathit{instOf}(pers, \mathit{Person}) \wedge && \text{(A1.2)}\\
&\quad \mathit{instOf}(proj, \mathit{Project}) \wedge && \text{(A1.3)}\\
&\quad \mathit{instOf}(top, \mathit{Topic}) \wedge && \text{(A1.4)}\\
&\quad \mathit{isAbout}(proj, top) \wedge && \text{(A1.5)}\\
&\quad \mathit{member}(pers, proj) && \text{(A1.6)}
\end{aligned}$$

(A1.1 to A1.6 means "A *Person* who works in a *Project* that is dealing with a *Topic*, *worksOn* this topic.")

Figure 2: Excerpt of SWRC ontology

The version we have used for our experiment comprises 199 concepts and 32 relations.

5. THE ASSEMBLY PROCESS

The assembly process comprises two main building blocks. First, the specification of temporal aspects for a time-less domain ontology remains dependent on the conceptualization of the ontology engineer. In fact, the example used for illustrating the assembly of general axioms below shows that there are ontological decisions to be made that can not be derived from the structure analysis of the two ontologies and therefore require human interaction. Second, in order to facilitate and accelerate the assembly of time-less domain concepts with temporal notions, the interactive process is supported by heuristics asking and pointing the engineer.

The assembly process runs as depicted in Figure 3: It starts by an Initial Setup. Some basic operations are performed, namely loading the ontologies to be assembled, loading a set of rules to drive the process and initializing some process parameters. The rules and parameters are defined separately from the tool in order to allow for adaptations to the particular needs of different temporal ontologies. However the rules and parameters do not change when a new domain ontology is to be assembled. The Target Ontology initially corresponds to the union of the time-less domain ontology, *O1*, and the temporal theory, *O2*.

Initially, the user may re-structure some part of the domain ontology to include temporal aspects by defining and executing (what we call) task instances. When performing

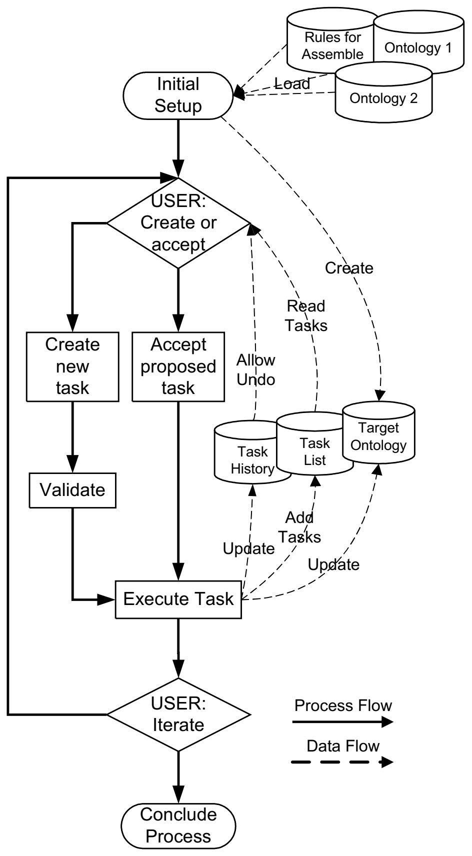


Figure 3: Assembly main process

such re-structuring task instances, a structure analysis finds possibly implicated task instances and proposes them onto the Task List. In subsequent iterations the engineer decides whether to accept an automatically proposed task instance from the Task List. Alternatively, the user may take new initiatives and define and execute a new task instance from scratch.

For manually defined task instances, a set of logical tests (Validate) are performed in order to detect the existence of any knowledge anomalies (e.g. circularity or redundancy [16]). In contrast, the acceptance of a proposed task instance does not require further checks as the checks are tested for validity before the user sees them.

By the Execute Task step the corresponding changes are made to the target ontology. Thereafter, the user decides either to pursue another iteration or to go to Conclude Process and accept the current Target Ontology as the final version.

5.1 Data Structures

Having outlined the principal procedure that operationalizes $\otimes$ and, hence, defines the result of assembling the time-less domain ontology with the temporal theory, we elaborate on the detail process in three subsections for assembling concepts ("Assembly of Concepts") , frame-like relations ("Assembly of Relations") and general axioms ("Assembly of General Axioms"), respectively. Before we can do so, we need to define the principal data structures we use to actually operationalize $\otimes$.

Task. We have already informally used the notion of *task* in order to refer to an action template (i.e. a generic task) that may be instantiated and executed in order to modify a current target ontology. A task is defined by its *task code* and a task question. The task code uses a set of keywords with the commonly expected semantics of structured programming (e.g. if, then, else) and some special keywords, do and propose, the semantics of which we provide subsequently.

Task instance. A task instance is fully identified by its head (i.e. by its task name and some instantiated arguments). A task instance is either proposed by the structure analysis or by instantiation of a generic task by the ontology engineer. For instance, consider the task instance

create-role-of(Student,Person),

which defines the concept *Student* to become a role of *Person*.

Task question. Before the execution of a task, the system asks a task question in natural language to the engineer in order to determine if the proposal should really be accepted or not and in order to ask for additional constraints that the user might want to add. The task question is defined by a List of words and parameters used to compose a sentence in natural language.

For instance, the following task question exists for the previous example

```
create-role-of(#arg1,#arg2):
['Define',#arg1,'as role of',#arg2,'?']
```

It implies that the question *"Define Student as role of Person?"* would be posed before executing the example task instance.

In order to manage various task instances, the assembling algorithm uses the following data structures:

Task List. This is a list of tuples
(TaskInstance,ListOfTriggers,Weight)
storing proposed task instances together with the triggers that raised their proposal and their weight according to which they are ranked on the task list. Thereby, *TaskInstance* has been defined before;

ListOfTriggers denotes the list of items that have triggered the proposal. A trigger is a pair *(TriggerType,TriggerId)* where *TriggerType* has one of the values *concept*, *relation* or *axiom* and the *TriggerId* is the item identifier. For instance, the pair *(concept, Person)* is a valid trigger. The list is useful to query for proposals raised by a specific item or *TriggerType.*

Weight. Since competing task instances may be proposed, *Weight* is used to reflect the strength of the proposal on the *TaskList*.

Task History is the list of all tasks that were previously performed. This list is useful to allow the undo operation and to provide statistics about the assembly process.

Task Constraints List. This is a list of tuples
(TaskInstance,Expression)
storing logical constraints about previously performed task instances. For instance:

```
constraint(
 create-role-of(S,C),
 NOT(
   performed(delete-relation(roleOf(S,C)))
   OR performed(create-relation(isa(S,C))))
)
```

means that the task create-role-of(S,C) should only be allowed, if neither such a role had been deleted before nor a competing subclass relationship (isa) had been created. Thereby, performed() acts as a lookup function into the Task History here. One may note that not all possible violations need to be stated here, as conventional KB validation mechanisms can be applied in order to detect undesirable task instances (e.g., ones that create isa-cycles although the ontology modelling policy forbids them).

do(TaskInstance). The function do performs logical tests over existing task constraints about *TaskInstance.* If there is no impediment it executes the task instance and creates a corresponding entry on the Task History.

propose(TaskInstance,Trigger,Weight). The function propose creates a proposal by asserting the corresponding tuple in the Task List.

5.2 Assembly of Concepts

As mentioned before, system proposals are generated based on rules and constraints. In the initial phase, the engineer takes the initiative. From the initial modifications, some first proposals may be generated automatically, and from these new proposals are spawned. Furthermore, the assembly of concepts with temporal attributes needs to fulfill fewer constraints than the assembly of relations and far less than the assembly of axioms. Thus, proposals for modifications with concepts are typically made first — and elaborated in this subsection.

For the running example here, we assume that a user defines and executes a task instance of assemble-concepts that subclasses a concept $C1$, viz. *person*, from a $C2$, viz. *TimedConcept*:

```
task assemble-concepts(C1,C2)
   if (C2='TimedConcept' or C2='Role')
       do(create-relation(isa(C1,C2)))
       assemble-related-concepts(C1)
       assemble-related-relations(C1)
       assemble-related-axioms(C1)
   end-if
end-task
```

The corresponding task assemble-concepts creates a new *isa* relation between the *Person* and *TimedConcept* and then proposes further assembling tasks for related concepts, relations and axioms. Tracing the changes that may be proposed to related concepts in assemble-related-concepts[2], we find that it proposes the definition of *Employee*, *Student* and *Male* as possible roles of *Person*.

```
task assemble-related-concepts(C)
  foreach S do
    %check if S is a sub-concept of C
    if(isa(S,C))
     %T is the trigger for this proposal
      T=(concept,C)
      % W is the weight foreseen
      % for this specific task
      propose(create-role-of(S,C),T,W)
    end-if
  end-do
end-task
```

Later, if definition of *Student* as role becomes accepted, recursively *Graduate* will be proposed to become a role of *Student* utilizing create-role-of:

```
task create-role-of(S,C)
%delete isa(S,C) if true; if not, no effect
    do(delete-relation(isa(S,C)))
    do(create-relation(roleOf(S,C)))
    assert(temporal-role-constraint(S,C))
    do(assemble-concepts(S,'Role'))
end-task
```

with temporal-role-constraint(S,C) defined by

$$\begin{aligned}\forall o, p1, p2 : containedBy(p1, p2) \leftarrow \\ instOfDuringAt(o, S, p1) \wedge \\ instOfDuringAt(o, C, p2). \qquad \text{(C1.1)}\end{aligned}$$

Assuming that *Male* were not accepted as role of *Person* in the further course of assembly, the result depicted in Figure 4 would be obtained.

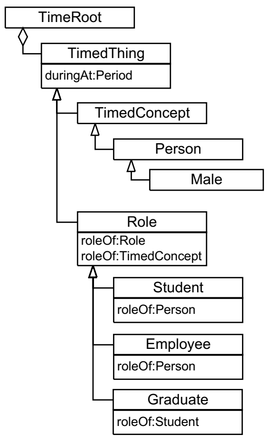
The reader may note that this result crucially depends on our temporal theory, but that rules could be easily modified to accommodate other theories (e.g., ones without roles).

5.3 Assembly of Relations

From the assembly of concepts there follow proposals for the modification of relations. For instance, when we assume that *Person* and *Project* were previously modified to become subconcepts of *TimedConcept* and *Process* respectively, it becomes plausible that also the relation that links them, viz. *member*(*Person*, *Project*), should incur changes.

The changes occur in analogy to the tasks defined for the assembly of concepts. In addition however, there arise further possibilities in order to constrain the life-time of the actual relationship by the life-time of the participating concept instances. Thus, *member*(*Person*, *Project*) is replaced by *member*(*Person*, *Project*, *Period*) and — maybe — further constraints on the time period as added by the engineer.

[2]The character '%' indicates a line remark.



$isa(Person, TimedConcept)$
$isa(Male, Person)$
$isa(Student, Role)$
$isa(Employee, Role)$
$isa(Graduate, Role)$
$roleOf(Student, Person)$
$roleOf(Employee, Person)$
$roleOf(Graduate, Student)$

$$\forall o, p1, p2 : containedBy(p1, p2) \leftarrow \\ instOfDuringAt(o, \mathrm{Graduate}, p1) \wedge \\ instOfDuringAt(o, \mathrm{Student}, p2).$$

Figure 4: Excerpt of Result of Assembly of Concepts with Time

In the running example, the proposed temporal modification of relation *member* is accepted while the one for *isAbout* isn't (e.g., because the engineer wanted to abstract from intricacies at this point). One may note that the rejected proposal appears to be conceptually 'nicer' than what the engineer decided to have in the end.

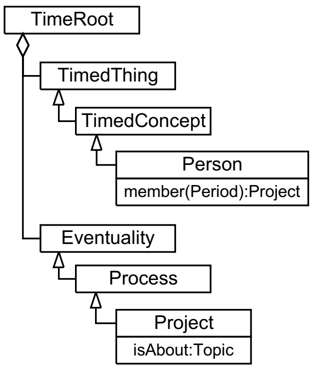


Figure 5: Excerpt of Result of Assembling Relations

5.4 Assembly of General Axioms

The temporal constraints on concepts, relations and their instances also requires the corresponding consistent modifications of general axioms. With general axioms we here refer to general propositions in first-order horn logics with function symbols.

For instance, let us consider the axiom defined in Figure 2 by lines A1.1 through A1.6 and let us name it *axiomWorksOn*. *axiomWorksOn* defines that a *Person* who works in a *Project* that is dealing with a *Topic*, *worksOn* this topic.

In order to assemble time into the axiom representation we must consider the constraints available for the instances of participating concepts and, furthermore, the ontology engineer must define which one of these constraints is used in which way. For instance, for the relation *worksOn* it may be adequate to say that the *Person worksOn* the *Topic* as long as he is a *member* of the *Project* and as long as the *Project isAbout* the *Topic*, i.e. intersecting the lifetimes of relations in lines A1.5 and A1.6. For an analogous structure where *knowsAbout*($pers$,top) appears in the head instead of *worksOn*($pers$,top), however, the conclusion might be that a *Person knowsAbout* a *Topic* ever after he has encountered it in a *Project* until he dies, i.e. restraining the *knowsAbout*-relation only to the earliest time-point of the encounter and the life-time of the *Person*.

Since, the only difference between the two example structures lies in the naming of the relations and the intentions associated with their names, it is necessary to involve the user for defining additional temporal constraints.

The task assemble-axiom (cf. Figure 6) asserts an additional temporal precondition for each concept that is temporally quantified. In our running example, this affects the instances of *Person* (A1.2) and *Project* (A1.3). Furthermore, it also updates the preconditions of the axiom involving relations that need to be temporally modified. In the running example, this affects, e.g., *member* (A1.6). Finally, the user is asked to define one (or several) constraint(s) that relates all the timed variables (intervals or instants) for the pre- or the postconditions of the axiom.

Then the updated axiom includes previously existing, but partially modified, preconditions and a further purely temporal one (A2.6). The modified axiom looks as follows:

$$\begin{aligned} &\forall pers, top, t : \mathit{worksOn}(pers, top, t) \\ &\quad \leftarrow \exists prj, t1, t2, t3 : \\ &\quad \mathit{instOfDuringAt}(pers, \mathrm{Person}, t1) \wedge && \text{(A2.1)} \\ &\quad \mathit{instOfDuringAt}(prj, \mathrm{Project}, t2) \wedge && \text{(A2.2)} \\ &\quad \mathit{instOf}(top, \mathit{Topic}) \wedge && \text{(A2.3)} \\ &\quad \mathit{isAbout}(prj, top) \wedge && \text{(A2.4)} \\ &\quad \mathit{member}(pers, prj, t3) \wedge && \text{(A2.5)} \\ &\quad \mathit{tempRelation}([t1, t2, t3], t) && \text{(A2.6)} \end{aligned}$$

with *tempRelation* being confirmed by the user to conform to the standard proposal:

$$tempRelation \equiv t \subseteq \bigcap_{i=1\ldots3} t_i \qquad \text{(A2.7)}$$

6. USER INTERACTION

FONTE uses an approach that relies on iterative interaction with the user. All the assemble operations as well as interactions with the *Task List* and the *History List* can be performed using a command line interface [3].

[3]For further information on user interface please refer: `http://www.dei.isep.ipp.pt/~jsantos/FONTE`

```
task assemble-axiom(Axiom)
 % an Axiom have an Head (single tuple)
 % and a Body (list of constraints)
 % LV is a list of timed variables

 %Process timed concepts of Body
 foreach instOf(C,T) partOf Body
  %non-instantaneous concepts
  if isa(T,'Process') or
     isa(T,'TimedThing')
  then add-constraint(during(C,TimeVar))
       LV=concat(LV,TimeVar)
  end-if
  %instantaneous concepts
  if isa(T,'Event') or
  then add-constraint(atTime(C,TimeVar))
       LV=concat(LV,TimeVar)
  end-if
 end-do

 %Process timed relations of Body
 foreach R(C1,C2) partOf Body
  if isa(R,'TimedRelation')
  then replace-constraint(R(C1,C2,TimeVar))
       LV=concat(LV,TimeVar)
  end-if
 end-do

 %select a temporal constraint over
 % all the timed variables and
  if LV not empty
  then add-temporal-constraint(A,LV)
       %update axiom head
       replace-head()
  end-if
end-task
```

Figure 6: The Task Definition for 'assemble-axiom'

In order to evaluate the effectiveness of FONTE, we have numerically evaluated the assembly tasks proposed and executed for the two case studies presented before. An ontology engineer interactively assembled the SWRC and the temporal ontology as well as the soccer ontology and the temporal ontology. Through our process log we could count the number of tasks proposed by the engineer and by the system as well as the number of tasks accepted by the engineer. The corresponding numbers are listed in Tables 1 and 2, respectively:

User-initiated tasks	10
Tasks proposed on *Task List*	135
Proposed tasks accepted	78
Proposed tasks postponed for later inspection	10
Proposed tasks rejected	47

Table 1: Summary for SWRC assemble

With regard to the SWRC assembly task, the engineer initiated only very few tasks (10). From this initial structure a large number of tasks were proposed (135). Many of these (58%) were accepted (78), 7.4% (10) were postponed for later inspection and the rest was ignored. I.e. only 34.8% were considered inadequate indicating a high success rate for our interactive approach.

User-initiated tasks	11
Tasks proposed on *Task List*	85
Proposed tasks accepted	67
Proposed tasks postponed for later inspection	10
Proposed tasks rejected	8

Table 2: Summary for Soccer assemble

This does not only seem to be true for simple structures (where it might have been expected), but also for modifications of horn-logic axioms. Originally SWRC contained 25 axioms. 20 of them were updated automatically the rest needed some more careful inspection.

For the second case study, achieved results were even somewhat better as only 9.4% (8) of 85 proposed tasks were rejected and 78.8% of proposed tasks were immediately accepted by the ontology engineer. The soccer ontology came without general horn logic axioms, thus we cannot make a statement about the performance of our method on these more intricate structures.

7. RELATED WORK

In the past a variety of approaches were proposed for reducing the complexity of engineering a rule-based system, e.g. by task analysis [19], or an ontology-based system, e.g. by developing with patterns [5, 21, 10] or developing subontologies and merging them [15, 17]. As different as these methods are, they may be characterized by subdividing the task of building a large ontology by engineering, re-using and then connecting smaller parts of the overall ontology.

Though FONTE shares its goal with these methodologies is its rather different in its operationalization. FONTE does not aim at a partitioning and re-union (by merge or align with recognition of similarities) of the problem space, but rather by a factorization into primordial concepts and a subsequent combination $\otimes$ that is more akin to a Cartesian product than a union of ontologies. Despite the difference, one may note that FONTE implements an iterative and interactive approach which was previously successfully adopted in sophisticated tools for merging ontologies [15, 13, 12]. Also, FONTE does not substitute these other methodologies, rather we envision that one wants to separate the target ontology to be built into different (possibly overlapping) domains *as well as* into time-less and temporal subontologies. The two ways of carving up the engineering task need different, complementary methodologies.

There is a rich set of languages and systems that deal with intricate reasoning over time and objects (cf., e.g., temporal Description Logics [2], temporal databases [3], or event calculus [4]). To our knowledge, however, there has not been a methodology that helped the ontology engineer to build ontologies that included temporal theories.

8. DISCUSSIONS AND FUTURE WORK

We have proposed a method, named FONTE, for engineering complex ontologies with time by assembling them from time-less domain ontologies and a temporal theory. FONTE provides an operator $\otimes$ that operationalizes the assembly in an interactive way. $\otimes$ combines two independently developed ontologies into a target ontology.

Finally, we here want to weigh advantages and disadvantages of the way we have presented the assembly operator $\otimes$. Certainly, the disadvantage of stating the assembly problem in this way is that the reader might expect a more formal and closed definition of the problem than one can give, as $\otimes$ is essentially based on heuristics and human conceptualizations. We see this as a small disadvantage as it 'just' concerns the management of expectations, what $\otimes$ can perform now.

For the future, however, we see the fruitful possibility in progressing the analysis that Wiederhold [26] has initiated. Wiederhold considered the problem of composing ontologies, the solution of which is similarly based on concept names, human intuition, and heuristics. He stated the solution of the problem in terms of operators (i.e. intersection '$O_1 \cap O_2$', subtraction '$O_1 \setminus O_2$', and union '$O_1 \cup O_2$') that may fulfill all the conditions for an algebra. Thus, one might harvest all the benefits of an algebra. In our case, this means our methodology might be complemented by projection operators π_1, π_2 to invert the way of assembly ($\pi_1(O_1 \otimes O_2) = O_1$), by operators to add spatial theories ($(O_1 \otimes O_2) \otimes O_3$), or by interactions with Wiederholds operators to juggle with different ontologies at will (e.g., compute $O_1 \otimes (O_2 \cup O_3)$).

To sum up, we have only studied the assembly of time into a given ontology so far, but we have reason to conjecture, *(i)*, that FONTE may also be applied to integrate other important concepts like space, trust, or user access rights — concepts that pervade a given ontology in intricate ways and necessitate management of engineering complexity; and, *(ii)*, building on Wiederhold's idea of an ontology algebra, the proposed operator $\otimes$ might perhaps be further utilized for having ontologies interoperate in manifold ways.

Acknowledgments

This work was partially supported by a Marie-Curie Fellowship and by FCT (Portuguese Science and Technology Foundation) with programs ONTOMAPPER (POSI-41818) and SANSKI (POCTI-41830). We are also in indebted to our colleagues in LS3, particulary to Gerd Stumme and Julien Tane and Nuno Silva in ISEP.

9. REFERENCES

[1] J. Allen. Maintaining knowledge about temporal intervals. *Communication ACM*, 26(11):832–843, 1983.

[2] A. Artale and E. Franconi. A temporal description logic for reasoning about actions and plans. *Journal of Artificial Intelligence Research*, 9:463–506, 1998.

[3] C. Bettini, X. S. Wang, and S. Jajodia. Temporal semantic assumptions and their use in databases. *IEEE Transactions on Knowledge and Data Engineering*, 10(2):277–296, 1998.

[4] L. Chittaro and A. Montanari. Efficient temporal reasoning in the cached event calculus. *Computational Intelligence*, 12:359–382, 1996.

[5] P. Clark, J. Thompson, and B. Porter. Knowledge patterns. In *KR2000*, pages 591–600, 2000.

[6] C. Freksa. Temporal reasoning based on semi-intervals. *Artificial Intelligence*, 54(1):199–227, 1992.

[7] N. Guarino and C. Welty. A formal ontology of properties. In *Knowledge Acquisition, Modeling and Management*, pages 97–112, 2000.

[8] P. Hayes, F. Lehmann, and C. Welty. Endurantism and perdurantism: An ongoing debate. http://ontology.teknowledge.com:8080/rsigma/dialog-3d-4d.html, 2002.

[9] J. Hobbs. Towards an ontology for time for the semantic web. In *Proc. Workshop on Annotation Standards for Temporal Information in Natural Language,LREC2002*, Las Palmas, Spain, May 2002.

[10] C.-S. J. Hou, N. F. Noy, and M. A. Musen. A template-based approach toward acquisition of logical sentences. In *Procs.Intelligent Information Processing 2002 - World Computer Congress*, Montreal, Canada, 2002.

[11] F. Lehmann. Big posets of participants and thematic roles. In *Conceptual Structures: Knowledge Representation as Interlingua*, pages 50–74. Springer-Verlag, 1996.

[12] D. McGuinness, R. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontologies. In *Procs.KR2000*, 2000.

[13] P. Mitra and G. Wiederhold. An algebra for semantic interoperability of information sources. In *Proc.2nd.IEEE Symp. on BioInformatics and Bioengineering, BIBE 2001*, pages 174–182, Bethesda, MD, 2001.

[14] I. Niles and A. Pease. Toward a standard upper ontology. In *Procs of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*, 2001.

[15] N. Noy and M. Musen. Prompt: Algorithm and tool for automated ontology merging and alignment. In *Procs.AAAI-2000*, 2000.

[16] A. Preece and R. Shinghal. Foundation and application of knowledge base verification. *International Journal of Intelligent Systems*, 9(8):683–702, 1994.

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

[18] J. Santos, C. Ramos, Z. Vale, and A. Marques. Verification & validation of power systems control centres kbs. In *Procs.IASTED Artificial Intelligence and Applications (AIA2001)*, pages 324–329, Marbella, Spain, September 2001.

[19] G. Schreiber, A. H., A. A., R. Hoog, N. Shadbolt, W. Van de Velde, and B. Wielinga. *Knowledge engineering and management. The CommonKADS Methodology*. MIT Press, 1999.

[20] J. Sowa. Processes and participants. In *Conceptual Structures: Knowledge Representation as Interlingua*, pages 1–22. Springer-Verlag, 1996.

[21] S. Staab, M. Erdmann, and A. Maedche. Engineering ontologies using semantic patterns. In *Procs. IJCAI-01 Workshop on E-Business & the Intelligent Web*, 2001.

[22] S. Staab and A. Maedche. Knowledge portals: Ontologies at work. *AI Magazine*, 22(2):63–75, 2001.

[23] F. Steimann. On the representation of roles in object-oriented and conceptual modelling. *Data & Knowledge Engineering*, 35(1):83–106, 2000.

[24] L. Vila and E. Schwalb. A theory of time and temporal incidence based on instants and periods. *Proc. Int. Workshop on Temporal Representation and Reasoning*, pages 21–28, 1996.

[25] M. Vilain and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proc. of AAAI-86*, pages 377–382, Philadelphia, PA, 1986.

[26] G. Wiederhold. An algebra for ontology composition. In *Proc.Workshop on Formal Methods*, pages 56–61, Monterey, 1994.