Engineering a complex ontology with time

Jorge Santos
GECAD - Knowledge Engineering and
Decision Support Research Group
Instituto Superior de Engenharia do Porto
Departamento de Engenharia Informática
4200-072 Porto - Portugal

Steffen Staab
Universität Karlsruhe (TH)
Institut AIFB
D-76128 Karlsruhe - Germany

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.

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 [**?**] and power systems [Santos *et al.*, 2001], 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.

[0]Jorge Santos was supported by a Marie-Curie Fellowship for a research visit to University of Karlsruhe. Most of the work presented here, has been developed during this research stay.

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 that includes time into two building blocks, viz. 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 build by assembling 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 [Noy and Musen, 2000; Rahm and Bernstein, 2001]. 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 [Wiederhold, 1994] 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 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.

$\otimes$ 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 opera-

tor $\otimes$ to a case study, where *O1* is a time-less ontology about a semantic web research community and *O2* is a temporal ontology. We have investigated how many assembling steps were proposed and evaluated their adequacy. The study results suggest that indeed FONTE provided a way to factorize the complexity of building large applications leading to more reliable and cheaper final products.

The rest of the paper is organized as follows. In Sections 2 and 3, we sketch the temporal ontology and the time-less domain ontology we have used for our case study, respectively. In Section 4 we describe the semi-automatic process of assembling two ontologies by $\otimes$, with some emphasis on the tool support developed to drive the process. Then we give an evaluation of our sample case in Section 5.

2 Temporal Ontology

The temporal ontology used for our case study (see Figure 1 for the depiction of an excerpt) embodies many concepts like *Instant* or *Period* routinely found in 'standard' ontologies like Time-DAML [Hobbs, 2002] or SUMO [Niles and Pease, 2001]. As argued by [Vila and Schwalb, 1996] 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., [Lehmann, 1996; Sowa, 1996; Guarino and Welty, 2000; Steimann, 2000]).[1]

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. [Hayes *et al.*, 2002]). The scope of this paper is to show a factorization of complexity into two sub-ontologies and a re-assembly into a target ontology in a way that is independent of such underlying concerns.

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 symbols:$\prec$, $\succ$, $=$, allowing to define an algebra based on points [Vilain and Kautz, 1986]. 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 [Allen, 1983] can be defined in a straightforward way based on the previous three relations [Freksa, 1992].

[1]The reader may note that we could not directly use a 'standard' temporal ontology like Time-DAML or SUMO as they do not include the notion of *role*.

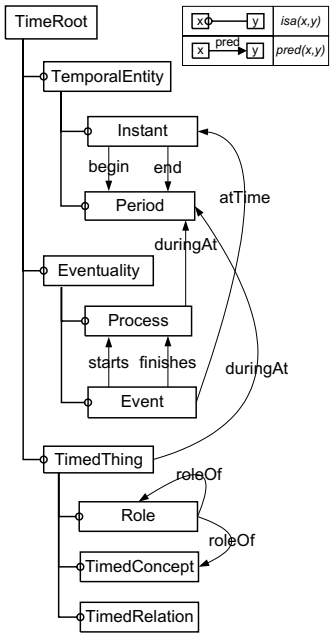

Figure 1: Excerpt of temporal ontology used in case study

While *begin* and *end* are relations from *TemporalEntity* to *Instant*, for convenience it is stated that *start* and *end* of an *Instant* is itself. 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* have a relation *atTime* to *Instant* while *Process* have a relation *duringAt* to *Period*. The relations *starts* and *finishes* allows to state what can start or finish one process, that relations can be useful when modelling a reasoning mechanism like Event Calculus [Kowalski and Sergot, 1986].

Roles. A role can have roles but can not be role of itself (R1) and that *roleOf* relation it is transitive (R2). Guarino distinguishes roles from natural types based on the lack of semantic rigidity of the first, hence, it is possible to derive that a role is timely bounded to the concept from which inherits identity (R3).

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

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

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

3 Case Study Example: The SWRC Ontology

The assemble process may be used either for development of ontologies with time from the scratch as well as for re-engineering existing ones in order to include time. For our case study we have used the 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 5 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 $\otimes$.

The SWRC ontology contains different types of axioms, namely, transitive, inverse, symmetric and general, being that, any axiom that does not fall in the first three types is called 'general'.

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

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

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

Figure 2: Excerpt of SWRC ontology

4 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. Therefore, it is very important that the engineer may interactively influence the process. 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 assembling 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*.

In the `Analyze Structure` step a set of tests are performed that restrict the set of possible task instances to plausible ones, which are then proposed by insertion into the `Task List`. As more information becomes available in

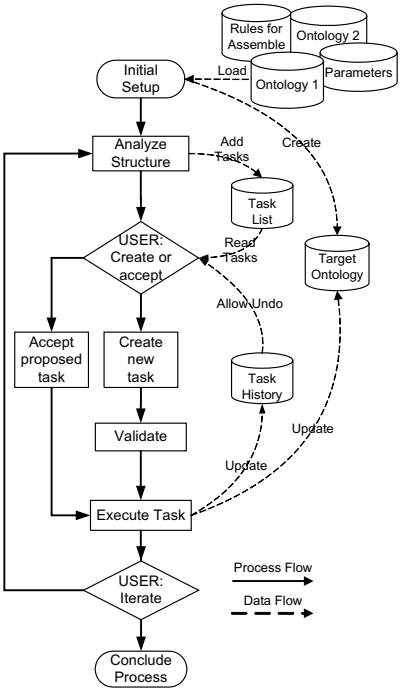

Figure 3: Assembly main process

subsequent iterations, the usefulness of results provided by the structure analysis improves.

In every iteration the engineer decides whether to accept an automatically proposed task instance from the `Task List`. Alternatively, the user may take the initiative and assemble a new task instance from scratch. Then a set of logical tests (`Validate`) are performed in order to detect the existence of any knowledge anomalies (e.g. circularity or redundancy [Preece and Shinghal, 1994]). 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.

4.1 Data Structures

In the remainder of Section 4, we first present the principal data structures we use for defining the heuristics, before we elaborate on its application in three subsections for assembling concepts, frame-like relations and axioms, respectively (Sections 4.2 to 4.4).

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 *procedure* 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 either proposed by the application of the assembly rules in the `Analyze Structure` step or by instantiation of a generic task of the ontology engineer (both ways operationalizing $\otimes$). 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 for queries about what proposals by a specific item or a certain *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.

`do(TaskInstance).` The function `do` executes a task instance and creates a corresponding entry on the `History List`.

`propose(TaskInstance,Trigger,Weight).` The function `propose` creates a proposal and asserts the corresponding tuple in the `Task List`.

4.2 Assembly of Concepts

As mentioned before system proposals are generated based on rules and constraints. Typically, in the initial phase only few interactions between the initially time-less domain ontology and the temporal theory are known and, hence, few proposals are generated. Furthermore, the assembling 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*:

```
procedure 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-procedure
```

The corresponding procedure `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`, we find that it proposes the definition of *Employee*, *Student* and *Male* as possible roles of *Person*.[2]

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

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

```
procedure 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-procedure
```

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

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

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).

[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*)

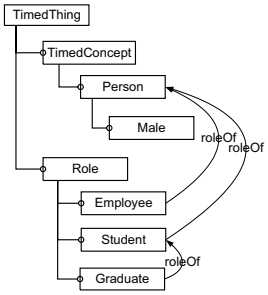

Figure 4: Result of assembly of concepts with time

4.3 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 contraints on the time period as added by the engineer.

4.4 Assembly of Axioms

The temporal constraints on concepts, relations and their instances also requires the corresponding consistent modifications of axioms. With 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` 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.

```
procedure assemble-axiom(A)
  %for each concept present in the axiom
  %LC is a list of all new constraints
  foreach C partOf(A)
    if isa(C,'Eventuality') or
       isa(C,'TimedThing')
    then add-concept-constraint(C,A)
       LC=concat(LC,C)
    end-if
  end-do
  %for each relation present in the axiom
  %LR is a list of modified constraints
  foreach R partOf(A)
    if isa(R,'TimedRelation')
    then modify-relation-constraint(R,A)
       LR=concat(LR,R)
    end-if
  end-do
  %select a temporal constraint over
  % all the timed variables
  if (LC not empty) or (LR not empty)
  then select-temporal-constraint(A,LC,LR)
  end-if
end-procedure
```

Then the updated axiom includes previously existing preconditions (A1.2 to A1.5) and further ones about *Person* (A2.1), *Project* (A2.2) and *member* (A2.3) — as well as a temporal constraint (e.g.intersection) over them (A2.4). Hence the structure of the modified axiom looks like:

$$\begin{aligned}
&\forall p, top, t : \textit{worksOn}(p, top, t) \leftarrow \\
&\quad \text{(A1.2)...(A1.5)} \\
&\quad \exists t1, t2, t3 : \textit{duringAt}(p, t1) \wedge && \text{(A2.1)} \\
&\quad \textit{duringAt}(prj, t1) \wedge && \text{(A2.2)} \\
&\quad \textit{member}(p, prj, t3) \wedge && \text{(A2.3)} \\
&\quad tempRelation([t1, t2, t3], t) && \text{(A2.4)}
\end{aligned}$$

5 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 both the graphical drag-and-drop tool or a command line interface (please refer to appendix A).

In order to evaluate the effectiveness of FONTE, we have numerically evaluated the assembly tasks proposed and executed for the ontologies presented in Sections 2 and 3. An ontology engineer interactively assembled the two ontologies and by our process log we could evaluate the success of our approach by listing the numbers from Table 1:

The reader may note that the engineer initiated only very few tasks (10). From this initial structure a large number of tasks were proposed and accepted by the user (135). Of these

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: Figures summarizing the log file

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.

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.

6 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 [Schreiber *et al.*, 1999], or an ontology-based system, e.g. by developing with patterns [Clark *et al.*, 2000; Staab *et al.*, 2001; Hou *et al.*, 2002] or developing subontologies and merging them [Noy and Musen, 2000; Rahm and Bernstein, 2001]. 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 these difference, one may note that FONTE implements an iterative and interactive approach which was previously successfully adopted in sophisticated tools for merging ontologies [Noy and Musen, 2000; Mitra and Wiederhold, 2001; McGuinness *et al.*, 2000]. 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.

7 Conclusions and Future Work

We have proposed a method, named FONTE, for engineering complex ontologies with time by assembling them from timeless 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.

Though, so far, we have only studied the assembly of time into a given ontology, we conjecture 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 such that a method like FONTE is needed in order to factorize engineering complexity out leading to more consistent and cheaper target ontologies.

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.

References

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

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

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

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

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

[Hobbs, 2002] 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.

[Hou *et al.*, 2002] Chih-Sheng Johnson 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.

[Kowalski and Sergot, 1986] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[Lehmann, 1996] F. Lehmann. Big posets of participants and thematic roles. In G. Ellis & G. Mann In P.W. Eklund, editor, *Conceptual Structures: Knowledge Representation as Interlingua*, pages 50–74. Springer-Verlag, 1996.

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

[Mitra and Wiederhold, 2001] 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.

[Niles and Pease, 2001] 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.

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

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

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

[Santos *et al.*, 2001] 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.

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

[Sowa, 1996] J. Sowa. Processes and participants. In G. Ellis & G. Mann (Eds.) In P.W. Eklund, editor, *Conceptual Structures: Knowledge Representation as Interlingua*, pages 1–22. Springer-Verlag, 1996.

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

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

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

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

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

A User's Choices

A command line interface was developed for the tool that implements the FONTE method. Table 2 describes commands used on the execution of assemble tasks both are available through the acceptance of automatic produced proposals or by user driven operations.

Command	Description
`link(C1,C2)`	Assembles concept *C1* from domain ontology with concept *C2* from general ontology
`link(R,C1,C2)`	Defines the relation *R* between concepts *C1* and *C2* as a temporal relation
`acptnextprop`	Accepts the next proposal in the *Task List* for execution
`acptpropnum(N)`	Accepts proposal *N* from *Task List* for execution
`acptprops(L)`	Accepts a list of proposals *L* from *Task List* for execution
`undo`	Undo last performed assemble task

Table 2: Assemble Tasks

Table 3 describes commands used for handling *Task List* and *History List.* Provided facilities are showing and sorting list contents.

Command	Description
`shownextprop`	Show detailed information about next task on *Task List*, including a question in natural language
`showprop(N)`	Show detailed information about task *N* on *Task List*, including a question in natural language
`sortproplist(F)`	Sorts tasks in *Task List* by field *F* (e.g.*Weight* or *Trigger*)
`showproplist`	Shows content of *Task List*
`showhistlist`	Shows content of *History List*

Table 3: Historic and Task Lists Handling

The table 4 describes commands used for handling files used in the assemble process, namely, ontology files and batch files.

Command	Description
`loaddomont(F)`	Loads contents of file *F* containing a domain ontology
`loadgenont(F)`	Loads contents of file *F* containing a general ontology (e.g. time ontology)
`loadscript(F)`	Loads a batch file *File* with a assemble tasks
`savescript(F)`	Saves into a batch file *F* the list of user driven assemble tasks recorded in *History List*
`savetargont(F)`	Saves target ontology obtained through the assemble process into file *F*

Table 4: File Handling

Finally, table 5 describes some auxiliary commands.

Command	Description
`cmd(C)`	Performs command *C*, allowing to evoke any assemble task not available through command line
`help`	Shows help texts about available commands
`exit`	Finishes program execution

Table 5: Auxiliary Commands