A Knowledge Representation System based on a Content Access for Dialogical Interactions

Jean-Paul Sansonnet, Erika Valencia

LIMSI-CNRS Université Paris-Sud; 91403 Orsay, FRANCE*

Topic Area : Representational Formalisms

Abstract

We propose a Knowledge Representation System (KRS) based on a content access to handle dialogical interaction within the InterViews project of the LIMSI laboratory. This KRS is necessary to overcome the semantical heterogeneity that emerges from the interactivity that we tackle. This project studies the possible interactions between software components called views, in a dialogical mode. We propose a design language for the definition of such views that integrates from its very conception interaction and reasoning services, as well as dynamic aspects.

1 Introduction

1.1 Background and motivations

The encodinging hypothesis is a strong limitation to the study of knowledge representation [5]. Indeed, it appears that we must also consider the interaction processes that allow the construction of such representations. However, when introducing interactions between agents, we also introduce a semantical heterogeneity between them; thus we must handle this heterogeneity in the knowledge representation of each agent itself.

*{erika, jps}@limsi.fr

The study presented here takes place into the InterViews project of the LIMSI laboratory[1]. The aim of this project is to study the dialogical interactions between software components modeled as agents, and called Views. Let us define the key notions of *View* and *Dialogism*.

Definition : A dialogical interaction is defined as a subset of a natural language interaction; it is based on *queries* exchange.

The queries that we introduce are supposed to contain more information than those used in classical data bases, since they also encapsulate intentions, and probably in a future work, affects.

Definition : A view is defined as a basic software component, in other words, a coherent unit of representation and reasoning, with its own vocabulary, concepts, memory and laws.

This means that a particular view contains a definition of its structure and dynamical properties using its own vocabulary. Each view has the capability to interact via dialogical queries with orther views, but since they have been designed by different persons, at different moments and for different purposes, there emerges a semantic heterogeneity in each interaction.

Therefore, the direct consequences of our applicative field, are that the information is distributed over several views, and the views are themselves distributed; a semantical heterogeneity emerges between the views ; the knowledge representations of the views evolve dynamically.

To solve these constraints, we must integrate the different aspects into our KRS: dialogism and speech acts for the interactions management; knowledge representation for a semantical normalization; the representation have an evolution through time and space. In fact, the main idea in the InterViews project is to keep all these fields closely related to propose a KRS that integrates all of them ina robust and incremental way. In this paper, we will focus on the second aspect only : the semantical normalization,and we consider it through the specific mode of dialogical interactions. But when presenting the KRS, we have to keep in mind the other two aspects of its conception.

1.2 Semantical Heterogeneity

1.2.1 Weak Aliens Problem

When two views have different vocabulary and structure, we consider them as aliens to each other. The Weak Alien Problem concerns a dialogical interaction

[1] See http://www.limsi.fr/Individu/jps/interviews.htm

between two alien views. We say call this problem Weak because we make the important assumption that the views are embedded into the same environment when interacting. This has an important consequence in terms of the objects that they know (grounding) and the representation they construct about them [8].

The WAP is usually solved by assuming the existence of a common intermediate language that we will call here VDL (for View Design Language). Not assuming its existence would lead us to consider the *strong* version of the problem. As we said, VDL is the language used in InterViews for the definition of the views as wholes, with their structure dynamicity, interaction and reasoning capabilities.

1.2.2 Dialogical Queries

In the multi-agents community, we usually use communication formalisms inspired by the speech acts theory proposed by Austin and Searle. The dialogical interactions being a subset of a natural language interactions, we will use a weak implementation of these theories, using concepts found in languages such as KQML [2]. Thus, our basic dialogical queries are the classical : *ask*, *tell*, *achieve*.

In InterViews, each one of these queries carries one *access* followed by an action, also called its *interpretation*. The interpretation of an ask query is an information extraction, the interpretation of a tell query, is an information modification (put), and the interpretation of an achieve query is the execution of some part of (executable code) information.

In the following section, we study the different access modes that we need in order to overcome the heterogeneity problems for any query exchange.

1.2.3 Content Access

In a classical distributed system, the queries are handled by services that statically ensure the coherence. In the case of alien views, this is no longer possible. We propose to use content access to retrieve and deliver informations in the context of semantical heterogeneity.

The different possible content access mode, correspond either to a pattern matching (according to a similarity measure) or to a structural pairing, or even both; these different methods are very close to the first steps of an analogical reasoning[2] .

[2]For which we previously gave a possible modelization based on topological tool[9].

2 A KR System based on a Content Access

Classically, a knowledge representation system is composed of a representation formalism based on a mathematical tool and services allowing the manipulation and in particular the access to the knowledge base represented with the previous formalism.

2.1 Expressing a Concept

Befre choosing a formalism (or language) to represent what we call the *base*, we note that we can express a *concept* by two fundamental (and dual) ways: 1)in *intension* : we need the definition and the use of a symbol of which the manipulation will be essentially syntactic. This symbol always refers to a content; 2) in *extension* : it is the developed expression of the properties, attributes or exemplary representatives that participate into the definition of a content.

From now on, we will keep both expressions, and note each concept belonging to a view, with both its extension and intension:

$$\langle T\rangle\ [C]$$

where $\langle T\rangle$ stands for the intension, and corresponds to a *ype* in the programming languages terminology, and $[C]$ stands for its extension, that we will call its *body*.

2.1.1 Relation "is-a"

The types of all the concepts of a view form an ontology, with the *is*-a relation [6]. Thus, each view possesses its own ontology, and even if we can find the same type in two views, there is no reason that it has the same is-a relations with other types in each view. In the actual model, the is-a relations are represented within simplicial complexes, which we have already shown are very well adapted to represent hierarchies and typologies [9][3].

2.1.2 Relation "part-of"

The extensions are defined recursively with other concepts, and thus can contain other intension/extension couples. The extensions included in the definition of a concept are related to it by a part-of relation [6]. The part-of structure of a view,

[3] And that offer some equivamences with the description logics [1], and the conceptual graphs [7].

is represented as a tree. We can study several different part-of relations [3] in a view, that lead us to colored trees..

2.2 A Syntax for the Views

2.2.1 Definitions

A view Υ is a software component corresponding to the basic granularity of our knowledge representation system. It possesses itself an intension, the type $\langle V \rangle$, and an extension recursively defined as follow:

$vue ::= \langle V \rangle [term];$

$term ::= \langle type^{*} \rangle [\{term\}] | termnial;$

where $type$ is a symbol belonging to the vocabulary A of the view. The notation $type^{*}$stands for *one or more elements,* the notation $a|b$ means that either the constructions a or b are acceptable. Finally, $\{term\}$ means a set of terms.

For each viewΥ, let Abe the set of symbols T_i of its vocabulary, then for each type $\langle T_i \rangle$, we introduce three kinds of attributes (types in its content):

- *compulsory* : for the description of the concept, they are sufficient to recognize one type (notion of prototype);
- *optional* : accepted but not necessary;
- *forbidden* : enough to reject it as belonging to the type.

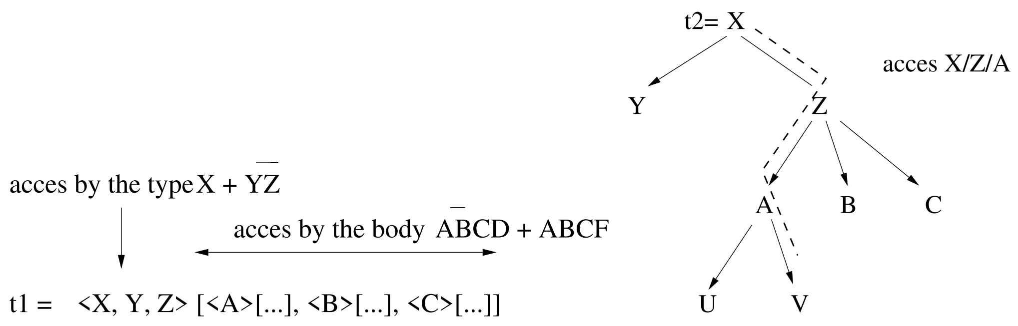
For each T_i, these three subsets resp. $A(T_i)^+$, $A(T_i)^o$ et $A(T_i)^-$ form a partitioning of the set A. We will see the link between these definitions and the use of *prototypes* [6].

2.3 Fundamental Access Modes

As we mentioned it before, the semantical heterogeneity, imposes that each access to the knowledge representation is done using the explicit expression of the concepts (from the point of view of the view accessed). Since we may use either the type or the body of a concept, to access to it, we define the following fundamental access modes:

- τ-access : by the type,
- κ-access : by the body,

Figure 1: τ, κ and π-access modes.



- π-access : along paths containing either τ-access or κ-access.

2.3.1 Access Syntax

The syntax for a τ-access mode is given below:

$\tau ::= \langle exprconj\rangle|\langle exprconj + exprconj^*\rangle;$

$exprconj ::= expr|expr.expr^*;$

$expr ::= type|\overline{type}$;

The$+$operator stands for a *inclusive* or, and the . operator for a *logic*al and. Moreover, the . operator has a higher priority. If a $type$ is present in the expression of an access, it means it is compulsory, if we mention its negation (noted $\overline{type}$), it means that it is forbidden, otherwise it is just optional.

The syntax of a κ access mode is given below:

$\kappa ::= [exprconj]|[exprconj + exprconj^*];$

$exprconj ::= expr|expr.expr^*;$

$expr ::= type|\overline{type}$;

And finally, the complete syntax of a π-access mode is:

$\pi ::= nodexpr|nodexpr/nodexpr^*;$

$nodexpr ::= \tau|\kappa;$

2.3.2 Algorithm and Semantics

The expression of the access mode is not enough to completely define them. Indeed, the specificity of these accesses is that they use pattern matching over the content, this implies a particular algorithm for each one of them. Here is the general algorithm to perform a π access mode; it is given by the Φ function that uses the Φ_τ and Φ_κ functions whenever it encounters a τ or a κ access mode:

$\Phi : (\Upsilon, \pi) \to \Upsilon$

$\Phi(\Upsilon, \pi, \Upsilon') ::=$

If $\pi \neq \emptyset$,

Let $\pi_o = First(\pi)$

Let $\gamma_o = Parcours(\Upsilon)$

If $\tau - acces(\pi_o)$ update $(\Upsilon', \Phi_\tau(\gamma_o, \pi_o, \Upsilon))$

If $\kappa - acces(\pi_o)$ update $(\Upsilon', \Phi_\kappa(\gamma_o, \pi_{o,} \Upsilon))$

Else $Fin(\Upsilon')$

$\Phi(\gamma_o, Reste(\pi, \pi_o), \Upsilon')$

where Υ' is the view returned by Φ. $First(\pi)$ takes the first elementary access $nodexpr$ in π and $Reste(\pi, \pi o)$ keeps the rest. The functions $\tau - acces(\pi_o)$ and $\kappa - acces(\pi_o)$ test whether π_o is resp. a τ or a κ mode. The function $Fin(\Upsilon')$ handles the errors whenever Φ terminates without any component in Υ' , and returns the empty view; this happens if $\pi \neq \emptyset$, or if π_o is not a τ or a κ mode. Finally, the function $Parcours(\Upsilon)$ is supposed to implement a classical tree exploration algorithm.

The functions Φ_τ and Φ_κ are local accesses applied to the current concept γ_o of Υ to which is accessing the $nodexpr$ π_o, either on the type, or on the body. Let $\gamma_o = \langle t_o \rangle [c_o]$, where t_o is the set of all the types of γ_o, then the function Φ_τ is defined by:

$\Phi_\tau : (\gamma_o, \pi_o, \Upsilon) \to \gamma_o$

$\Phi_\tau(\gamma_o, \pi_o, \Upsilon) ::=$

If $\mu_\tau(t_o, \pi_o, \Upsilon)$ returns γ_o,

Else $Fin(\Upsilon')$

It returns the concept γ_o for its updating in Υ' if its internal matching function μ_τ returns a success, this last function is defined by:

$\mu_\tau : (t_o, \pi_o, \Upsilon) \to \{True, False\}$

$\mu_\tau(t_o, \pi_o, \Upsilon) ::=$

If $\pi_o \in t_o$ then $True$

Else If $subsume(\pi_o, t_o, \Upsilon)$ then $True$

Else $False$

where the function $\in$tests a *symbolic membership* and the $subsume(\pi_o, t_o, \Upsilon)$ function tests the *existence* of a is-a link between π_o and one of the elements of t_o, it uses for this purpose the ontology of Υ.

The function Φ_κ is defined according to the same rules:

$\Phi_\kappa : (\gamma_o, \pi_o, \Upsilon) \rightarrow \gamma_o$
$\Phi_\kappa(\gamma_o, \pi_o, \Upsilon) ::=$
If $\mu_\kappa(c_o, \pi_o, \Upsilon)$ returns γ_o,
Else $Fin(\Upsilon')$
but with the following matching function, μ_κ:
$\mu_\kappa : (c_o, \pi_o, \Upsilon) \rightarrow \{True, False\}$
$\mu_\kappa(c_o, \pi_o, \Upsilon) ::=$
If $\pi_o inc_o$ then $True$
Else If $contient(\pi_o, c_o, \Upsilon)$ then $True$
Else $False$

where the in function tests if the $nodexpr$ π_o *matches* with one of the types of the set c_o and the $contient(\pi_o, c_o, \Upsilon)$ function tests whether π_o is one of the *parts* of c_o in the part-of tree of Υ. This last function allows *relative* π-accesses to be performed, that is paths in the view that do not give each step of access from the head of the view.

Note that the four functions $\in$, in, $subsume$, $contient$ used in the matching functions use the compulsory, forbidden and optional attributes as mentioned earlier. For example, if we look for a small man, into the description $\langle Human \rangle [small]$, it is correct to return a success if we posses the information man is-a human in the ontology of the view.

2.4 Dialogical Interaction

In the section 1.2.2 we proposed to model the dialogical interactions in Interviews starting from *queries*.

Definition : A query R is a couple (π, I) of a π-access and an interpretation.

This notion of interpretation finds its true meaning through the notion of *observation* of a view by another.

2.4.1 Notion of Observation

Definition : An observation Ω is a triplet (Ω, R, Υ), where Ω and Υ are two views and where Ω sends a query R to Υ.

We say then that Ω observes Υ or that Ω is the observer and Υ the observed view. In principle, any view can perform an observation, which corresponds to a query formulated with the vocabulary corresponding to its own definition (and thus its own body). But we also provide special views, that are privileged observers, their type is then $\langle \Omega \rangle$. These special observers, correspond to a special vocabulary (or keywords) that we may want to use to perform general observations over any view, such as a display or an execution. Indeed, the effect of a query being sometimes an extraction of information, we may want to have a special observer that contains all the vocabulary necessary for a general display (of the displayable parts) of all the views.

Definition : An observer Ω is a view where the content is seen as composed of keywords type $\langle Key\rangle[a]$ and a part $\langle Observer\rangle[b]$. where Key and $Observer$ are special types, a is a set of keywords, and b is a set of process that we wish to run during the observation.

$\Omega ::= \langle V\rangle[\langle Key\rangle[a], \langle Observer\rangle[b]]$

The standard observer that performs the *displaying* of the views, is called Ω_{show} :

$\Omega_{show} ::= \langle V\rangle[\langle Key\rangle[a_{show}]\langle Observer\rangle[b_{show}]]$

a_{show} is a set of only *spatial* keywords corresponding to displayable types of objects of InterViews.

We provide a second standard observer, called Ω_{exec} , that performs the ignition of the *dynamical* (executable code) parts of each view:

$\Omega_{exec} ::= \langle V\rangle[\langle Key\rangle[a_{exec}]\langle Observer\rangle[b_{exec}]]$

where $[a_{exec}]$ is the vocabulary necessary to tag[4] executable code of the view.

2.4.2 Observation Algorithms

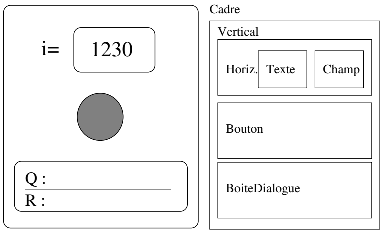
An observation is thus a general access to all the parts of a view, followed by an interpretation depending on the body of the observer. The algorithm necessary to perform an observation will then use the π-access mode for each observable part of the view.

$\langle Observer\rangle[\{\pi\}_{\langle Key\rangle}, \Im]$

$\{\pi\}_{\langle Key\rangle}$ stands for all the possible π-accesses using the $\langle Key\rangle[a]$ vocabulary and $\Im$ means either an extraction, a modification, or an execution. If the query is an ask, then $\Im$ simply corresponds to extraction processes and the whole observation is just a set of π-accesses as defined by the Φ function (in 2.3.2). If the query is a tell, $\Im$ we can add adaptation processes to the formation of the answer, by induction [4]. This step is in fact the key part for the resolution of the WAP in InterViews, we propose an adaptation algorithm for it highly inspired of our preceding topological model for analogical reasoning; this peace of work is not presented here.

[4]This notion os close to a public declaration of type.

Figure 2: Visualization of the counter on the screen, and corresponding spatial structure.



Note that the expression $\langle X\rangle[\langle Y\rangle[a,b], \langle Z\rangle[c]]$, assumes that the term $\langle Y\rangle[a,b]$ takes the position of the attribute a which is compulsory for the definition of type X. Thus, the fact of knowing exactly which attribute is compulsory for each type, will allow its identification.

2.5 Application

We give now a complete example of view, from the definition, to the standard observations on it.

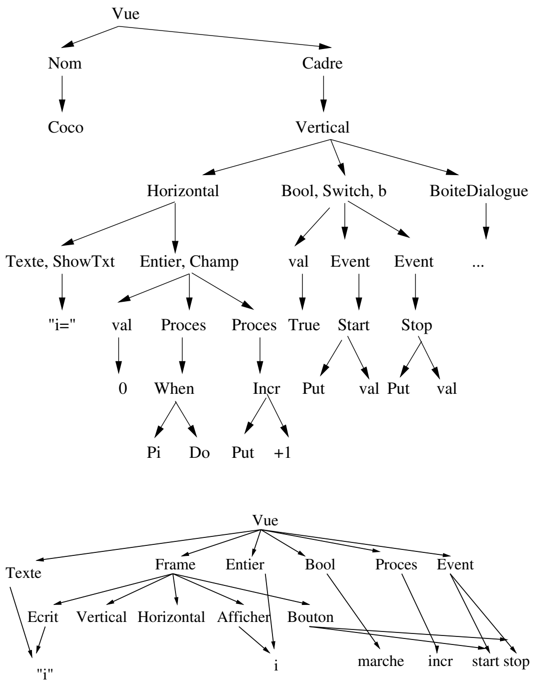
2.5.1 Designing a Counter

The view we design here is a counter, with a start/stop button, that can be displayable on screen. This view contains : an integer for the current value of the counter, a display for this value, a procedure to increment it, one to stop it, and one to initialize it, and finally a name "Coco". The figure 2, on the left, shows its vizualisation on screen.

The spatial structure of the counter, shown on figure 2 right, implies the use of *spatial types* to the design of the corresponding view. Let $\langle Vertical\rangle$ stand for a vertical alignment, $\langle Horizontal\rangle$, for a horizontal one, and $\langle Frame\rangle$ to contain any kind of alignments. The definition of Coco into the VDL language will thus be:

$\langle V\rangle[\langle Nom\rangle[Coco],$
$\langle Frame\rangle[\langle Vertical\rangle[$
$\langle Horizontal\rangle[$
$\langle Texte, Ecrit\rangle[i],$
$\langle Entier, Affiche\rangle[\langle i\rangle[0]],$

Figure 3: Part-of structure of Coco, and its corresponding ontology.



$\langle Bool\rangle[\langle marche\rangle[True]]]$
$\langle Horizontal\rangle[$
$\langle Processus\rangle[...\langle inc\rangle[i]...],$
$\langle Event, bouton\rangle[..\langle stop/start\rangle[marche]...]]]]$

Where ... stand for executable code. Since VDL has been implemented with the Mathematica [10] programming language, the executable code is directly written using the Mathematica syntax.

2.5.2 Ontology and Structure

The structure of the body of Coco, in terms of part-of relation is given by the tree and the induced ontology of types are given by the figure 3.

This ontology is not a tree, but a complex, but we preferred to sketch it as a Hasse diagram, since there is an equivalence between them (see [9]).

3 Further Work and Conclusions

The knowledge representation system that we propose is an integrated design language, called VDL, that handles interactivity through queries, semantical heterogeneity through content access, and dynamicity and evolution through the integration of executable code into the design of the views. Thus a view is not a static representation that could be updated, but also an operating world with its own procedures and laws. The interaction part of VDL is now being improved through the definition of a little specialized module of VDL, called VQL. The queries are enriched and include affect and modalities.

We also study in more details the problems brought up by the semantical heterogeneity. Indeed, the content access alone is not capable of solving all the problems that emerges from this heterogeneity, since it does not deal with terms that do not exist at all into the body of the view addressed. In particular, the WAP must be solved by the use of more sophisticated *protocols* of interaction allowing the exchange of more information to match a term by *adaptation* from an example. This proposition again uses our preceding studies concerning analogy and we already proposed an algorithm for such an adaptation, assuming that the ontologies are expressed with simplicial complexes.

Finally, more complex examples have been programmed, like the remote control of a micro-wave oven through the web (see web page of the project). opefully, these examples will bring up to light more and more subjects of analysis and help us improving the VDL language.

References

[1] F. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. *Principles of Knowledge Representation*, chapter Reasoning in Description Logics, pages 193–238. Studies in Logic. CSLI Publications, 1996.

[2] Tim finin and Richard Fritzson. KQML as an agent communication language. In *CIKM'94*. ACM Press, November 1994.

[3] Nicola Guarino. Formal ontology, conceptual analysis and nowledge representation. *IJHCS*, 43(5/6), 1995.

[4] John H. Holland, Keith J. Holyoak, Richard E. Nisbett, and Paul R. Thagard. *Induction – Processes of Inference, learning, and Discovery*. The MIT Press, 1986.

[5] Clayton T. Morrison. Are we wrong about representations ? *JETAI*, 9:441–469, 1997.

[6] Gérard Sabah. *L'intelligence Artificielle et le langage, la représentation des connaissances*. Hermès, Paris, 1990.

[7] J. F. Sowa. *Conceptual structures*. Addison-Wesley, 1984.

[8] Luc Steels and Frédéric Kaplan. *Linguistic evolution through language acquisition : formal and computational models*, chapter Bootstrapping grounded word semantic. Cambridge university press, 1999.

[9] Erika Valencia and Jean-Louis Giavitto. Algebraic topology for knowledge representation in analogy solving. In ECCAI, editor, *European Conference on Artificial Intelligence (ECAI98)*, pages 88–92, Brighton, UK, 23–28 August 1998. Christian Rauscher.

[10] Stephen Wolfram. *Mathematica*. Addison-Wesley, Redwood City, CA, 1988.