Visual and Textual Knowledge Representation in DESIRE

Catholijn M. Jonker[1], Rob Kremer[2], Pim van Leeuwen[1], Dong Pan[2], Jan Treur[1]

[1] Vrije Universiteit Amsterdam, Department of Artificial Intelligence
De Boelelaan 1081a, 1081 HV, Amsterdam, The Netherlands
Email: {jonker, treur}@cs.vu.nl, URL: http://www.cs.vu.nl/{~jonker,~treur}

[2] University of Calgary, Software Engineering Research Network
2500 University Drive NW, Calgary, Alberta T2N 1N4, Canada
Email: {kremer, pand}@cpsc.ucalgary.ca

Abstract. In this paper, graphical, conceptual graph-based representations for knowledge structures in the compositional development method DESIRE for knowledge-based and multi-agent systems are presented, together with a graphical editor based on the Constraint Graph environment. Moreover, a translator is described which translates these graphical representations to textual representations in DESIRE. The strength of the combined environment is a powerful -- yet easy-to-use -- framework to support the development of knowledge based and multi-agent systems. Finally, a mapping is presented from DESIRE, that is based on order sorted predicate logic, to Conceptual Graphs.

1 Introduction

Most languages for knowledge acquisition, elicitation, and reasoning result in specifications in pure text format. Textual representation is easier for a computer program to process. However, textual representation is not an easily understandable form, especially for those domain experts who are not familiar with computer programming. Visual representation of knowledge relies on graphics rather than text. Visual representations are more understandable and transparent than textual representations [7].

DESIRE (DEsign and Specification of Interacting REasoning components) [2] is a compositional [3] development method used for the design of knowledge-based and multi-agent systems. DESIRE supports designers during the entire design process: from knowledge acquisition to automated prototype generation. DESIRE uses composition of processes and of knowledge composition to enhance transparency of the system and the knowledge used therein.

Originally, a textual knowledge representation language was used in DESIRE that is based on order sorted predicate logic. Recently, as a continuation of the work presented in [6] a graphical representation method for knowledge structures has been developed, based on conceptual graphs [8].

Constraint Graphs [5] is a concept mapping "meta-language" that allows the visual definition of any number of target concept mapping languages. Once a target language is defined (for example, the DESIRE's graphical representation language) the

constraint graphs program can emulate a graphical editor for the language as though it were custom build for the target language. This "custom" graphical editor can prevent the user from making syntactically illegal constructs and dynamically constraints the choices of the user to those allowed by the syntax. Constraint Graph's graphical environment is used to present knowledge in a way that corresponds closely to the graphical representation language for knowledge that is used in DESIRE. A translator is described that bridges the gap between the graphical representation and the textual representation language in DESIRE.

Another well-known knowledge representation language, that also makes use of graphical notations, is Conceptual Graphs [8]. Knowledge presented in Conceptual Graphs can also be represented in predicate logic. Since DESIRE is based on order sorted predicate logic, such knowledge can also be represented in DESIRE. In this paper, a mapping is given from DESIRE to Conceptual Graphs, thus bringing DESIRE closer to that representation language.

2 Graphical Knowledge Representation in DESIRE

In this section both graphical and textual representations and their relations are presented for the specification of knowledge structures in DESIRE [2]. Knowledge structures in DESIRE consist of information types and knowledge bases. In Sections 2.1 and 2.2 graphical and textual representations of information types are discussed. In Section 2.3 representations of knowledge bases are discussed.

2.1 Basic Concepts in Information Types

Information types provide the ontology for the languages used in components of the system, knowledge bases and information links between components. In information type specifications the following concepts are used: sorts, sub-sorts, objects, relations, functions, references, and meta-descriptions. For the graphical specification of information types, the icons in Fig. 1 are used.
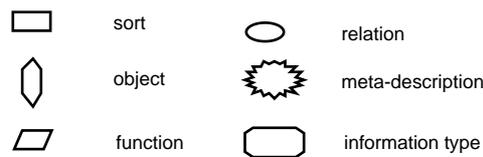
Fig. 1. Information types: legenda

A sort can be viewed as a representation of a part of the domain. The set of sorts categorizes the objects and terms of the domain into groups. All objects used in a specification have to be typed, i.e., assigned to a sort. Terms are either objects, variables, or function applications. Each term belongs to a certain sort. The specification of a function consists of a name and information regarding the sorts that form the domain and the sort that forms the co-domain of the function. The function name in combination with instantiated function arguments forms a term. The term is

of the sort that forms the co-domain of the function. Relations are the concepts needed to make statements. Relations are defined on a list of arguments that belong to certain sorts. If the list is empty, the relation is a nullary relation, also called a propositional atom. The information type birds is an example information type specifying sorts, objects, functions and atoms with which some knowledge concerning birds can be specified. The information type is specified in Fig. 2. With information type birds it is, for example, possible to express statements like "Tweety is of the type that it prefers vegetarian food": is_of_type(tweety, food_preference(vegetarian)).


```
information type birds
sorts                          BIRD,
                               DIET,
                               BIRD_TYPE;
objects
    tweety :                   BIRD;
    carnivore,
    omnivore,
    vegetarian :               DIET;
functions
    food_preference :     DIET--> BIRD_TYPE;
relations
    can_fly:                   BIRD;
    is_of_type :        BIRD * BIRD_TYPE;
end information type
```
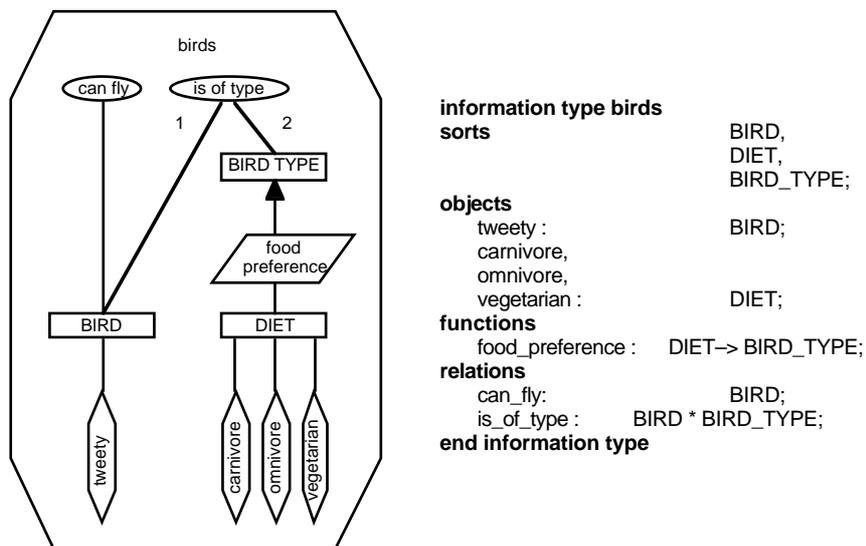
Fig. 2. Information type: **birds**

Note that being able to express a statement does not mean that the statement is true, it could be false.

2.2 Compositionality of Information Types

Compositionality of knowledge structures is important for the transparency and reusability of specifications. In DESIRE two features enable compositionality with respect to information types: information type references, and meta-descriptions. By means of information type references it is possible to import one (or more) information type(s) into another. For example, information type birds above can be used in an information type that specifies an extended language for specifying knowledge that compares birds.

Example 1
```
information type compare_birds
    information types   birds;
        relations            same_type:                          BIRD * BIRD;
    end information type
```

The second feature supporting compositional design of information types is the meta-description representation facility. The value of distinguishing meta-level knowledge from object level knowledge is well recognized. For meta-level reasoning a meta-language needs to be specified. It is possible to specify information types that describe the meta-language of already existing languages. As an example, a meta-information type, called about_birds, is constructed using a meta-description of the information type birds (see Fig. 3). The meta-description of information type birds connected to sort BIRD_ATOM ensures that every atom of information type birds is available as a term of sort BIRD_ATOM. Using information type about_birds it is possible to express that it has to be discovered whether bird Tweety can fly (to_be_discovered(can_fly(tweety))).
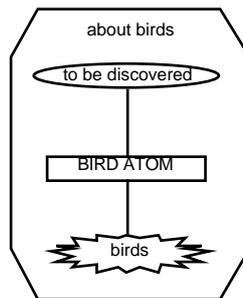
Fig. 3. Meta-descriptions: **about birds**

2.3 Knowledge Bases

Knowledge bases express relationships between, for example, domain specific concepts. Reasoning processes use these relationships to derive explicit additional information.

Example 2
```
knowledge base birds_kbs
    information types compare_birds;
    contents
        if              has_type(X: BIRD, Y: BIRD_TYPE)
          and           has_type(Z: BIRD, Y: BIRD_TYPE)
          then          same_type(X: BIRD, Z: BIRD);

        if              has_type(X: BIRD, type(Y: DIET, flying, Z: HABITAT))
          then          flies(X: BIRD);

        has_type(tweety, type(vegetarian, flying, hot));
    end knowledge base
```

The knowledge base birds kbs specified in Example 2 expresses which birds are of the same type, and which birds fly. Although, knowledge bases can be represented graphically as well, examples have been omitted from this paper.

Finally, a knowledge base can reference several other knowledge bases. The knowledge base elements of knowledge bases to which the specification refers are also used to deduce information (an example has been omitted).

3 Constraint Graphs

Constraint graphs is a concept mapping "meta-language" that allows one to visually define any number of target concept mapping languages. Once a target language is defined (for example, the DESIRE knowledge representation language) the constraint graphs program can emulate a graphical editor for the language as though it were custom build for the target language. This "custom" graphical editor can prevent the user making synactically illegal constructs. Furthermore, the editor dynamically constraints user choices to those allowed by the syntax.
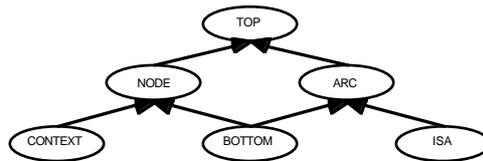
Fig. 4. The base type lattice for Constraint Graphs

In order to accommodate a large number of visual languages, constraint graphs must make as few assumptions about concept mapping languages as possible. To this end, constraint graphs defines only four base components: node, arc, context, and isa (see Fig. 4). Nodes and arcs are mutually exclusive, where nodes are the vertices from graph theory, and arcs interconnect other components, and are analogous to edges in graph theory. Both nodes and arcs may (or may not) be labeled, typed, and visual distinguished by color, shape, style, etc. Contexts are a sub-type of node and may contain a partition of the graph. Isa arcs are a sub-type of arc and are used by the system to define the sub-type relation: one defines one component to the be a sub-type of another component merely by drawing an isa arc from the sub-type to the supertype.

Futhermore, the generality requirement of constraint graphs dictates that arcs are not always binary, but may also be unary or of any arbitrary arity greater than 1 (i.e., trinary and n-ary arcs are allowed). For example, the between relation puts a trinary arc to good use. Constraint graphs arcs may interconnect not only nodes but other arcs as well. This is not only useful, but necessary because all sub-type and instance-of relations are defined using an isa arc, arcs between arcs are required to define the type of any arc. Finally, within constraint graphs no hard distinctions are made

between types and instances, but rather, the object-delegation model [1] is followed where any object can function as a class or type.
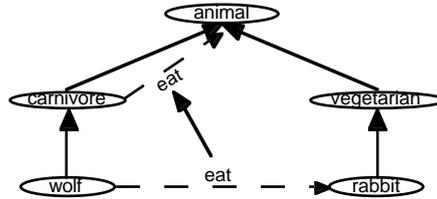
Fig. 5. An example Constraint Graphs definition

To illustrate some of the above points, Fig. 5 shows a simple definition. Here, the fat, directed arcs are the constraint graphs isa arcs and define carnivore and vegetarian to be sub-types of animal, wolf as a sub-type (or instance-of) of carnivore, and rabbit as a sub-type (or instance-of) of vegetarian. Furthermore the eat binary relation (dashed arc) is defined and starts on carnivore and terminates on animal. These terminals are important: the components at the terminals constrain all sub-types of eat to also terminate at some sub-type of carnivore and animal respectively. The second eat arc is defined (by the fat isa arc between it's label and the first eat arc's label) to be a sub-type of the first eat arc. It is therefore legally drawn between wolf and rabbit, but the editor would refuse to let it be drawn in the reverse direction: the eat definition says that rabbits can't eat wolves.

4 The Translator

In Constraint Graphs, three basic types of objects exist: nodes, arcs and contexts. The elements of the language to be expressed in the Constraint Graphs' environment therefore need to be mapped onto these basic types. Table 1 below shows the mapping between DESIRE's knowledge elements and nodes, arcs and contexts.

Object	Sort	Subsort	Meta description	Function	Relation	Information type	Knowledge Base
NODE	NODE	ARC	ARC	ARC	ARC	CONTEXT	CONTEXT

Table 1. Mapping between DESIRE and Contraint Graphs

Constraint Graphs allows the user to further constrain the language definition in by, for example, restricting the shapes and connector types of the nodes and arcs the language elements are mapped onto. In our case, we restrict the shape of node Sort to a rectangle, and the shape of Object to a diamond. Furthermore, sub-sorts, meta-descriptions and relations will be represented as directed labeled arcs, where the label takes the shape of an ellipse. Moreover, functions will be depicted as directed labeled

arcs as well, but the label will be a parallelogram. Finally, information types and knowledge bases are mapped onto contexts, and the shape of these contexts will be the default: a rectangle.

Fig. 6 below gives an impression of a specification of the DESIRE information type birds (compare to Fig. 2) in Constraint Graphs.
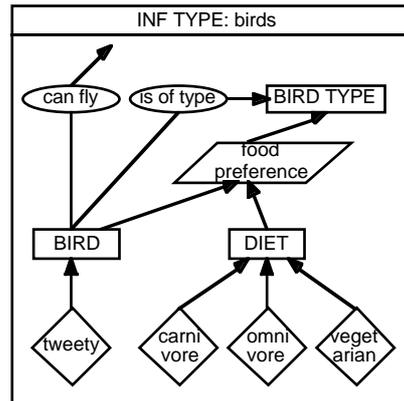
Fig. 6. Example of a DESIRE information type represented in Constraint Graphs

5 Relation to Conceptual Graphs

In this paper, graphical notations for knowledge in DESIRE are presented, as well as a translator which translates specifications of these notations in a graphical environment called Constraint Graphs to the textual DESIRE representation. Having this graphical interface brings the knowledge modelling in DESIRE closer to other well-known knowledge representation languages, such as Conceptual Graphs [8] because a dedicated interchange procedure could be added to the software. The relation between Conceptual Graphs and predicate logic is well-known. The fact that DESIRE is vased on order sorted predicate logic, and the possibility to represent different meta-levels of information within DESIRE, ensures that all knowledge represented in Conceptual Graphs can also be represented in DESIRE. In this section a translation from representations in DESIRE to representations in Conceptual Graphs is defined.

A conceptual graph is a finite, connected, bipartite graph, which consists of two kinds of nodes: concepts and conceptual relations. Concepts are denoted by a rectangle, with the name of the concept within this rectangle, and a conceptual relation is represented as an ellipse, with one or more arcs, each of which must be linked to some concept. Fig. 7 below shows an example conceptual graph, representing the episodic knowledge that a girl, Sue, is eating pie fast.
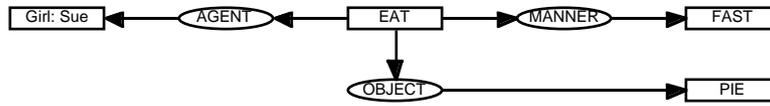
Fig. 7. An example Conceptual Graph

When comparing conceptual graphs with the graphical notations for DESIRE, many similarities become apparent. For instance, DESIRE's relations are denoted by ellipses, like conceptual relations, and sorts appear as rectangles, like concepts. Other elements however, are harder to translate to a Conceptual Graph notation. Table 2 provides an overview of the translation of DESIRE elements to Conceptual Graphs. Part of this is discussed in some detail.

Objects

Objects in DESIRE are instances of a sort. In Conceptual Graphs (CG) these instances are represented by individual concepts, i.e., concepts with an individual marker following the concept name. For example, the object tweety of sort BIRD in DESIRE is represented by [BIRD: tweety] in CG. Also anonymous individuals can be translated, e.g., the DESIRE variable X:BIRD is translated into [BIRD: *x] of CG which means that it is known that an individual of type BIRD exists, but it is unknown which individual.

Functions

In DESIRE, functions group sorts together by mapping them onto another sort. Functions can be regarded to be sub-types of a general CG concept FUNCTION, which takes one or more arguments and produces a result. In DESIRE functions act as a named placeholder for an object of its result, in which the argument(s) and the name of the function ensure the placeholder's uniqueness. Function food_preference, for example of Fig. 2, can be represented by the following Conceptual Graph:

    [DIET]<-(ARG)<-[food_preference]-(RSLT)-[BIRD_TYPE].

Relations

Relations in DESIRE can be classified according to their arity. This arity determines the mapping to Conceptual Graphs. 0-ary relations in DESIRE will have to be translated to concepts; concepts in Conceptual Graphs form a graph in itself, like nullary relations form a DESIRE atom in DESIRE. Relations with an arity greater than zero can be translated into either a conceptual relation with the same arity or a combination of a concept and (an)other conceptual relation(s). For example, the relation between: space * brick * brick in DESIRE could be translated into the following Conceptual Graph:

    [SPACE] - (BETW) - [BRICK]
                       -[BRICK]

This graph is a triadic relation, which could be read as "a space is between a brick and a brick". Relation is_traveling_from_to: person * origin * destination however could be translated into the graph

```
[TRAVEL]   -
           (AGNT) - [PERSON]
           (ORG) - [ORIGIN]
           (DEST) - [DESTINATION]
```

Sub-sorts

In DESIRE, hierarchical relations between sorts are allowed. Sub-sorts in DESIRE correspond to the type hierarchy of concepts in Conceptual Graphs. In Conceptual Graphs, hierarchies of both concepts and conceptual relations are possible, but these hierarchical is-a relations are kept in a separate semantic net from other relations that exist in the domain.

Desire Element	Graphical Equivalent in Constraint Graphs	Equivalent in Conceptual Graphs
Object	diamond	individual concept
Sort	rectangle	generic concept
Sub-sort	rectangle connected to super-sort by instance-of arrow	type hierarchy of concepts
Meta-Description	dashed arrow from information type to sort	conceptual relation -(METALEVEL)-
Function	parallelogram	concept FUNCTION
Relation	ellipse	conceptual relation or concept and conceptual relation(s)
Information type	context-box labeled SIG	context
Knowledge Base	context box labeled KB	context
Antecedent	context-box labeled ANT	context
Consequent	context-box labeled CONS	context
NOT-context	context-box labeled NOT	negative context
Information type Reference to information type	arrow between information types	context enclosed in another context
Knowledge base Reference to KB	arrow between knowledge base contexts	context enclosed in another context
KB reference to information type	arrow from kb to information type	comparable to first three and last component in a canon
Rule	arrow labeled "implies" between antecedent and consequent	conceptual relation -(IMP)-

Table 2. DESIRE, Constraint Graphs, and Conceptual Graphs

6 Conclusion

In this paper, graphical representations for knowledge structures in DESIRE [2] have been presented, together with a graphical editor based on the Constraint Graph environment [5]. Moreover, a translator has been described which translates these graphical representations to textual representations in DESIRE. This software

environment can be regarded as a graphical design tool for knowledge in DESIRE, an interface which offers many advantages to a textual interface. First, Constraint Graphs can be used to specify knowledge structures, allowing the user to work with a mouse, pull-down menu's and windows instead of typing the specification conform the textual DESIRE syntax. Second, the graphical representation of knowledge structures (supported by the software environment for Constraint Graphs) offers a clear visual representation, facilitating communication between domain expert and knowledge engineer in the development process. Third, the graphical representations bring DESIRE closer to other knowledge representation languages, such as Conceptual Graphs [8], by defining a mapping from DESIRE to Conceptual Graphs (the other direction was already covered). In conclusion, the strengths of the Constraint Graphs environment as an easy to use representation tool in combination with the DESIRE environment allows for a powerful framework to support the development of knowledge based or multi-agent systems.

References

1. Abadi, M., and Cardelli, L. *A Theory of Object*, Springer, New York, 1996.
2. Brazier, F.M.T., Dunin-Keplicz, B., Jennings, N.R., and Treur, J., Formal specification of Multi-Agent Systems: a real-world case. In: V. Lesser (Ed.), *Proceedings of the First International Conference on Multi-Agent Systems*, ICMAS-95, MIT Press, Cambridge, MA,, 1995, pp. 25-32. Extended version in: *International Journal of Cooperative Information Systems*, M. Huhns, M. Singh, (Eds.), special issue on Formal Methods in Cooperative Information Systems: Multi-Agent Systems, vol. 6, 1997, pp. 67-94.
3. Brazier, F.M.T., Jonker, C.M., and Treur, J., Principles of Compositional Multi-Agent System Development, In: J. Cuena (ed.), *Proceedings of the IFIP World Computer Congress*, WCC'98, Conference on Information Technologies and Knowledge Systems, IT&KNOWS'98, 1998.
4. Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, Mass., 1994.
5. Kremer, R., Constraint Graphs: *A Constraint Graphs Meta-Language*, PhD Dissertation, Department of Computer Science, University of Calgary, 1997.
6. Moeller J.U., and Willems M. CG-DESIRE: Formal Specification Using Conceptual Graphs; In: Gaines, B.R. and Musen, M.A. (eds), *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop* KAW-95, Calgary, 1995, pp. 25/1 - 25/20.
7. Nosek, J. T., and Roth, I., A Comparison of Formal Knowledge Representation Schemes as Communication Tools: Predicate Logic vs. Semantic Network, In: *International Journal of Man-Machine Studies*, vol. 33, 1990, pp. 227-239.
8. Sowa, J.F., *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, Reading, Mass., 1984.