Conceptual Graphs and Metamodeling

Olivier Gerbé[1], Guy W. Mineau[2], and Rudolf K. Keller[3]

[1] HEC Montreal.
3000, chemin de la Côte-Sainte-Catherine, Montréal, Québec, Canada H3T 2A7
Olivier.Gerbe@hec.ca
[2] Université Laval
Québec, Québec, Canada G1K 7P4
mineau@ift.ulaval.ca
[3] Université de Montréal
C.P. 6128 Succursale Centre-Ville, Montréal, Québec, Canada H3C 3J7
keller@IRO.UMontreal.ca

Abstract. Metamodeling is often identified as a key layer in the development of an information system because it formally defines the modeling primitives that will be used in subsequent modeling activities. We use the Conceptual Graph (CG) theory for illustration purposes. The simplicity of the CG notation and its flexibility to represent metalevel knowledge through the use of contexts makes it a serious contender for the representation of a metamodeling theory. Therefore, this paper presents a CG based metamodeling framework for the modeling of information systems.

1 Introduction

Metamodeling is often identified as a key layer in the development of an information system [2, 3] because it formally defines the modeling primitives that will be used in subsequent modeling activities. By defining the modeling language, the semantic constraints of the domain can be embedded into it, restricting the expressivity of the modeling language accordingly, ensuring greater consistency throughout the modeling of the domain. Also, queries concerning the modeling language itself can be answered. Using its formal definitions, the modeling language can be explained, which provides the essentials for the establishment of an on-line task support system. A reduction in the number of work hours spent to understand the modeling language, and therefore a gain in productivity, is sought. When many people act as knowledge modelers throughout some organization (as with consultant firms for instance), or when employee turn over is high, this gain in productivity is considerable [11].

Furthermore, through the use of these formal definitions, the modeling language can be validated. A valid modeling language, one in which all definitions are together compatible, i.e., do not produce any inconsistencies, improves the ability of the knowledge modeler(s) to produce a set of object definitions which are consistent with one another.

Finally, with a metamodeling approach the modeling language is defined in a declarative formalism. This allows partial or full mapping between different modeling languages, leading to systems integration. Therefore, system interoperability issues can be approached from a metamodeling point-of-view.

For all these reasons, we too advocate the use of a metamodeling layer in the development of an information system. We use the Conceptual Graph (CG) theory for illustration purposes. The simplicity of the CG notation and its flexibility to represent metalevel knowledge through the use of contexts makes it a serious contender for the representation of a metamodeling theory. Therefore, this paper[1] presents a CG based metamodeling framework for the modeling of information systems. Section 3 introduces the basic ontology required to develop a CG based metamodeling language. Section 4 presents a mapping function from the metalevel to the data level, allowing the definitions stated at the former level to be used at the latter level. Section 5 presents how this framework can be used to create an arbitrary number of metalevels. Section 6 introduces metarules that state restrictions and properties of metalevel definition primitives. Section 7 concludes and presents future directions for our research.

2 Literature Review

Metamodeling and conceptual graphs have not been extensively investigated. John Esch in [7] introduces metamodeling through two predefined relations: Kind that links a concept to its type and Subt that links two concept types that are in a subtype relationship. He defines, using the relationship Subt, a type hierarchy for each higher order level, and links, using relation Kind, types and concepts from different levels. But Esch does not deal with conceptual relations and relation types. In [26] Michel Wermelinguer defines more formally higher order types and proposes a translation to first order logic. He defines one hierarchy for all the concept types and one hierarchy for all the relation types and organizes them in regard of their nature and their order. Pavel Kocura in [12] deals with the semantics of attribute relations in conceptual graphs and introduces some second order concept types like: TYPE, REL_TYPE and relation types like ATTR and VALUE_TYPE. He also presents some mapping rules from higher level to lower level using Attribute (ATTR) relations. But none proposes a complete metamodel of the conceptual graph language itself.

3 Modeling Constructs in the CG Formalism

Through a mapping to first-order logic (FOL), the CG theory is recognized as a general knowledge representation language. The simple CGs which are fully mappable to FOL formulae are called first-order CGs. Additional features such as sets [21, 8], contexts [5, 6, 7, 16], and various quantifiers [1] provide higher

[1] This work is part of a research project supported by HEC Montreal.

reasoning capabilities by allowing modal [10], temporal [4, 19, 20] and fuzzy [1] reasoning systems to be devised based on the CG representation language.

First-order CGs are composed of concepts and relation nodes. Concepts represent objects (either physical or not) of some type. They are composed of the type of the object followed by a reference to the object that they represent, called a referent. Relations represent semantic links between objects. Relations are typed. The distinction between a concept and a relation is rather arbitrary. At times relations may be seen as objects. For simplicity purposes pertaining to both the modeling activities and the subsequent efficiency of the knowledge handling operators, the knowledge engineer must decide on a domain ontology that reflects the concepts of the domain and their possible relationships, all seen as primitive elements of the modeling language [14].

Section 3.1 introduces the representation primitives needed to describe concept types; Section 3.2 does the same with the definition of relation types. Together these sections provide the basic constructs needed to set up a metamodeling layer in a CG system.

3.1 Defining Concept Types

The definition of a concept type is about an object that is being defined (and specified) at the metalevel so that it can be used as a concept type at the data level. For example, let us introduce the object `[ConceptType: Driver]` which states that Driver is an object of type ConceptType (which is predefined). And let us use Driver in a concept `[Driver: *x]`. The former is useful to describe the properties of concept type Driver, therefore providing it with a formal definition. The latter is then permitted and can be used to describe individual drivers who will comply with the definition of concept type Driver as given by the former definition.

Therefore we first need a concept type ConceptType that is used to represent and define concept types. Then, when a concept type t is to be defined, we need to attach the corresponding concept type concept `[ConceptType: t]` to some definition graphs, represented by (embedded graphs) concepts, that will provide it with different roles and restrictions. These roles are indicated by the type of relation that links concept `[ConceptType: t]` to its definition graphs. The different relation types that are needed to define a concept type are: csubt, def, rstrct, and sntx; each links concept type concept `[ConceptType: t]` to a concept type concept, a definition graph, a restriction graph (or a rule graph), and a syntax graph, respectively. Each of these relation types is defined in the subsections below. For example, concept type Driver could be defined using the CG of Figure 1 asserted at the metalevel.

Subtyping: csubt

From Figure 1, the subtype relation Driver $<$ Person can be extracted from the csubt relation. The concept type hierarchy is therefore built from all such relations extracted from all concept type definitions. This creates an inheritance

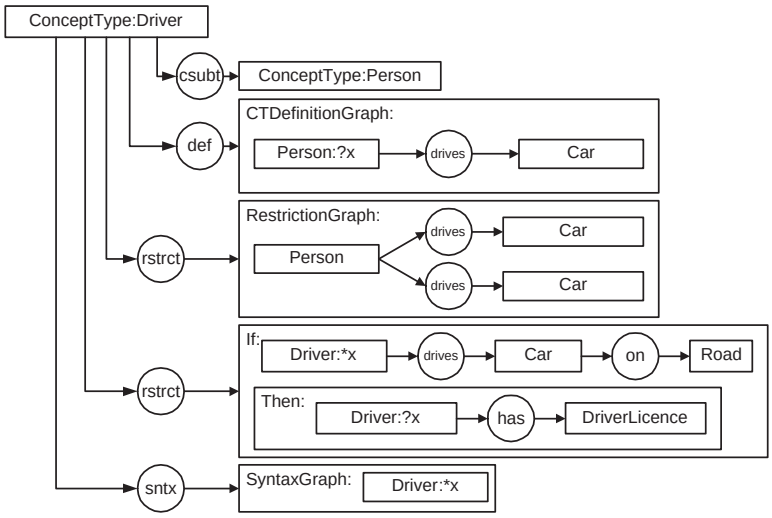

Fig. 1. An example of a concept type definition.

network among concept types where all linked pairs of concept types are part of a partial order of generality defined by the csubt relation.

Using a Concept Type at the Data Level: sntx

The syntax graph presents how the concept defined at the metalevel, a concept type in the case of Figure 1, must be used at the data level[2]. Here again, rules on how to compose a concept based on a concept type definition can be stated at the metalevel (see Section 6). These rules ensure that the syntactical forms are used according to the metalevel definitions on which the acquired objects are based.

Genus and Differentia: def

In Figure 1, it is stated that a driver is a person who drives a car. The genus of the definition is thus the concept of type Person; its differentia is the statement that s/he must drive a car to be recognized as a driver. This statement is equivalent to the lambda expression of Figure 2.

$$\text{Driver}(x_1) = [\text{Person: } \lambda x_1] \rightarrow (\text{drives}) \rightarrow [\text{Car}]$$

Fig. 2. The lambda expression extracted from the definition graph of Figure 1.

[2] The use of *x in the syntax graph and ?x in the definition graph is a lexical convention (see /refCGIG for semantics of *x and ?x). That does not mean they refer to the same individual except if there are in the same graph

Additional validation rules can be stated (as shown in Section 6) on how to ensure that the definition graph has a concept whose referent is ?x, and that the concept type of this concept is in accordance with the concept type which is the destination concept of the csubt relation.

Constraints on the Use of Types: rstrct

There are two types of restriction graphs, each one imposing a constraint on the use of the concept type being defined: restriction graphs and rule graphs. Restriction graphs introduce graphs that must never project themselves onto any other graph in a CG system[3]. Therefore, they are graphs that represent situations that must never occur. In our example of Figure 1, a driver cannot drive two cars at the same time. Rule graphs introduce complementary definitions to the main definition of a type, but only under certain conditions. With our example, when a driver drives a car that is on the road, then s/he must have a driver's license. That is, when the if-graph projects itself onto some graph in the CG system, then the then-graph must also project itself onto the same graph (providing that the coreferenced variables are bound to the same concepts). Restriction and rule graphs permit the representation of a large subset of the semantic constraints found in database literature. They were introduced under a slightly different representation in [18]. For a more complete introduction on restriction and rule graphs see [17] and [13]; for a formal definition of their associated extensional semantics see [15].

3.2 Defining Relation Types

As before, a predefined RelationType concept type is required to express that some object is a relation type. The primitive relations defined above either hold for the definition of relation types or have counter-parts. Figure 3 gives some example of the definition of a relation type.

Subtyping: rsubt

In Figure 3, relation type goingto is defined as a subtype of the Link[4] relation type. A relation type hierarchy can be built from the rsubt relations found in the definitions of all relation types. Here we chose to specialize the type inheritance relation (subt) for concept types (csubt) and relation types (rsubt). This choice, rather than using the subt relation directly, is justified for the following reasons. First, linked elements concept types and relation types are different. Second, the way to verify the validity of the relation is also different (See metarules in Figure 16 and Figure 18.)

[3] The projection that we consider here is injective. See [9] for the appropriate motivation. We believe that under certain simplifying assumptions (see [14]) an injective projection should be sought.

[4] The relation type Link is primitive and states a relationship between two concepts. Link is at the top of the relation type hierarchy.

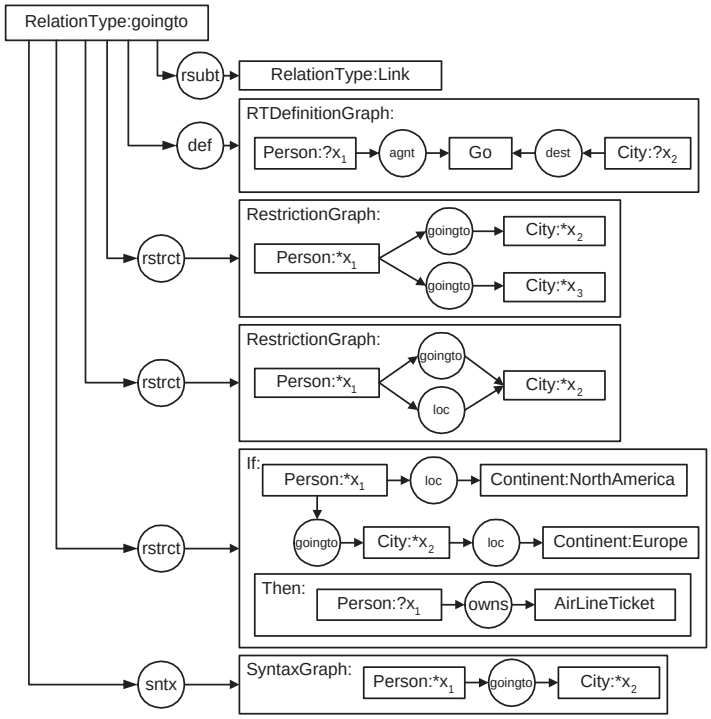

Fig. 3. The definition of a relation type..

Using a Relation type at the Data Level: sntx

Finally, the use of the relation type r at the data level must be represented. The syntax graph associated with the definition of r does that. As mentioned before, syntactical formation rules can be expressed at the metalevel in order to validate the use of a relation type at the data level (see Section 6).

Genus and Differentia: def

From Figure 3, one can see that the relation type goingto between two parameters x1 and x2, is defined as a person x1 who is the agent of a verb Go, for which the destination is a city x2. From this (relation type) definition graph, the lambda expression of Figure 4 could be extracted. Again, a metalevel rule can be expressed to verify that the definition of a relation type r conforms to the relation type of its supertype r' (according to the rsubt relation) (see Section 6).

$$\text{goingto}(x_1,x_2) = [\text{Person: } \lambda x_1] \rightarrow (\text{agnt}) \rightarrow [\text{Go}] \leftarrow (\text{dest}) \leftarrow [\text{Person: } \lambda x_2]$$

Fig. 4. The lambda expression extracted from the definition graph of Figure 3.

Using the lambda expression produced from the definition of a relation type r, the signature of r is then known. Therefore the canonical basis of the CG system, B, can be built from the analysis of the definition graph of each relation type. Also, a metarule enforcing the signature of each relation type can be expressed, as will also be presented in Section 6.

Constraints on the Use of Types: rstrct

As with concept type definitions, restrictions can be defined on the use of a relation type at the data level. For instance, Figure 3 states that a person cannot go to two different cities that s/he cannot go to a city where s/he is already located, and that when a person is located in North America and goes to a city in Europe, then s/he must own an airline ticket.

4 Mapping Metalevel to Data Level

In [9] we defined a function ω[5] that maps higher level to a lower level objects. Let us recall that definition.

Definition 1. *Function ω is defined over $\mathcal{C} \rightarrow \mathcal{E}$ where $\mathcal{C}$ is the set of concepts that represent entities of the system and $\mathcal{E}$ is the set of all referenced elements (internal and external elements[6]).*

Applied on a concept, function ω returns the entity represented by the concept. Obviously, the function is defined on the set of concepts that represent entities of the system, i.e., internal elements.

Figure 5 shows the way the function ω may be used. Let us have (a) graph `[City:Ottawa]->(cap)->[Country:Canada]` identified by the internal referent `#4387`, (b) two different ways to speak about this conceptual graph, and (c) the application of ω on the first concept of (b).

Figure 6 presents how the type of a concept could be accessed using a meta-level CG describing the concept. Let us consider `[Concept: [City:Ottawa]]` that is the concept that represents the concept `[City:Ottawa]`. Using the predefined `type` and `ref` relations and `Concept` and `Referent` concept types, let us have the graph of Figure 6.

Figure 7 shows the mapping from a higher level to its immediate lower level. Applying function ω on the concept of type `Concept` in the meta-level CG of Figure 6, we obtain the (data level) concept representing the city of Ottawa.

[5] The function ω is a generalization of the Sowa's functions τ and ρ[24]. ω is different of Sowa's function $referent$ that returns the lexical of the referent field[25].

[6] In the metamodel we distinguish two types of element, the external elements, external with the language, and the internal elements which are the components of the language. The external elements represent the objects of the universe of the speech which is outside the system and which can be referred by internal elements.

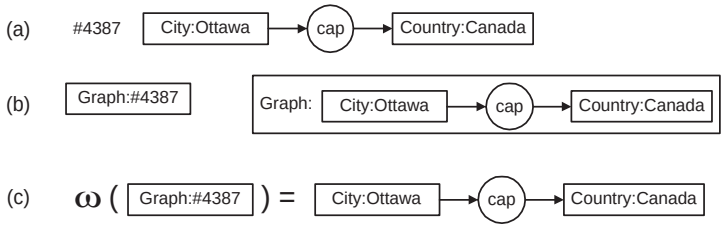

Fig. 5. Function ω.

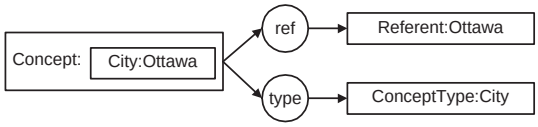

Fig. 6. From a lower level to a higher level.

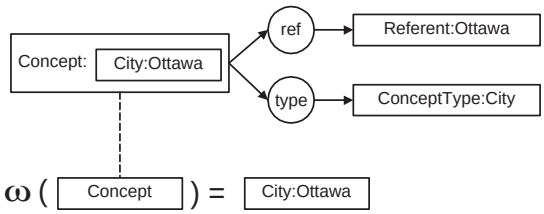

Fig. 7. From a higher level to a lower level.

5 Defining Subtypes of the Primitives of Section 3

The definition primitives of Section 3 and the mapping operator of Section 4 allow the objects of any level to be described by definitions found one level up, at their metalevel. This provides for many layers of modeling levels. One use for such layers is the definition of the modeling primitives from which the modeling language that we are describing in this paper is composed of, allowing different modeling languages to be mapped onto one another.

This section will illustrate these ideas by defining subtypes of certain relation types, creating classes of relations, thus specializing the modeling language even more (Section 5.1) and classes of specialized graphs (Section 5.2). By doing so, we aim at demonstrating how general the framework described in this paper is.

5.1 Creating Classes of Relations

The RelationType concept type, used as a primitive element in our modeling language ontology so far, is itself a concept type. Therefore, it could be defined using the definition primitives introduced in Section 3.1. Doing so will allow specializations of it to be defined, refining further the modeling language that will be handed out to the knowledge acquisition modules in charge of modeling the

actual application domain. In this section, we intend to show the expressivity of the simple representation tools introduced in Sections 3 and 4. As a first example, let us say that a relation type is a type that is subtype of another relation type. Figure 8 below illustrates this definition. Notice that there is no syntax graph associated with it since a syntax graph represents a precise syntactical form, therefore, a predetermined and fixed number of parameters (the arity of relations of that type) would be required.

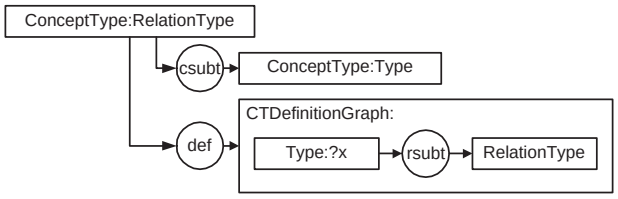

Fig. 8. The definition of a relation type.

Therefore, syntactic considerations lead us to define fixed-arity relation types. Figure 9 shows a specialization of concept type RelationType, BinaryRelationType, which will be useful for defining binary relations[7]. It imposes a particular syntax graph to all of its elements, providing a fixed arity of two for all relations of that type.

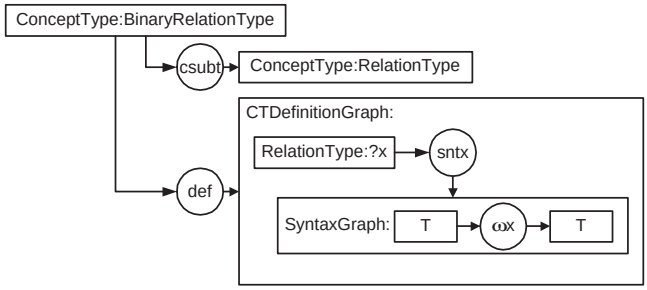

Fig. 9. The definition of a relation type for binary relations.

Other subclasses of relation types can be defined in the same way. For instance, it is possible to define a class of transitive relations through the definition of a subclass of BinaryRelationType. Figure 10 gives such a definition.

Other classes of relations can be defined to match particular properties of relations, like symmetry (see Figure 11), anti-symmetry (see Figure 12), and reflexivity (see Figure 13).

[7] As introduced in [9] in order to simplify the notation we replace ω(`[RelationType:*r]`) by ω`r`

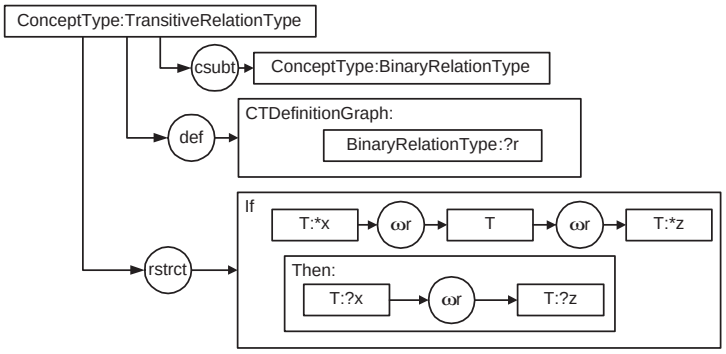

Fig. 10. The definition of a class (type) for all transitive relations.

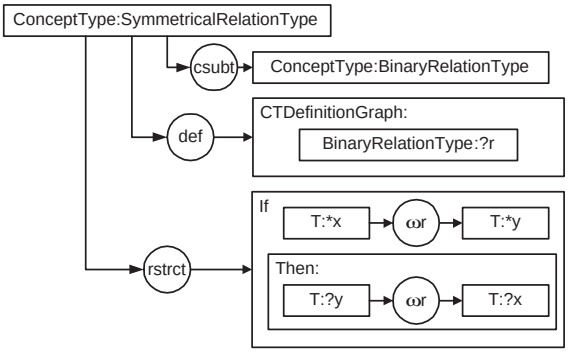

Fig. 11. The definition of a class (type) for symmetrical relations.

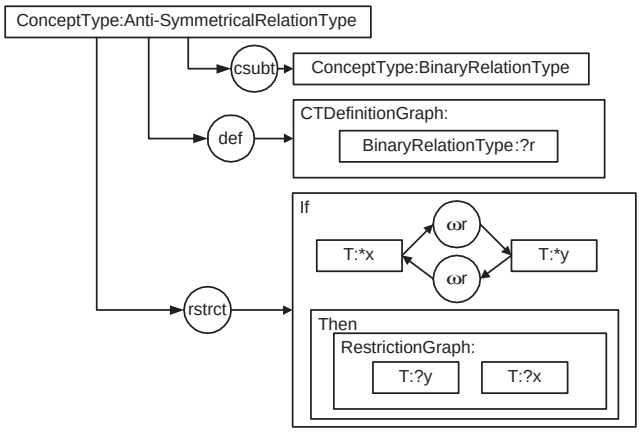

Fig. 12. The definition of a class (type) for anti-symmetrical relations.

In Figure 12 the restriction graph states that if x is in relation r with y and y is in relation r with x then x and y may not be two distinct concepts.

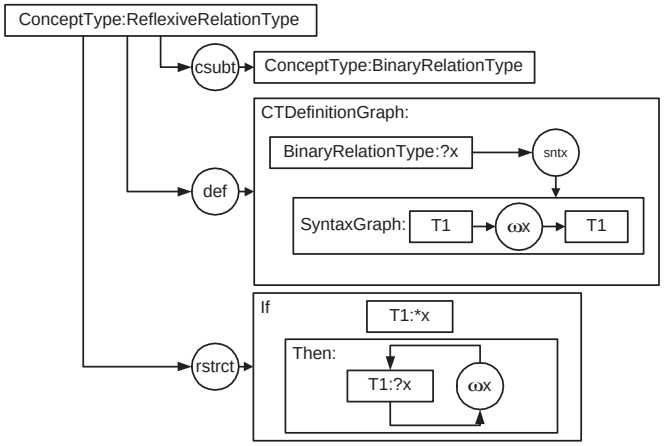

Fig. 13. The definition of a class (type) for reflexive relations.

In Figure 13 we need to use the syntactic graph to identify in the rule graph the type that the relation may link.

5.2 Creating Classes of Graphs

The CTDefinitionGraph concept type, used as a primitive element so far, may be defined as a concept type. A Concept Type Definition Graph (CTDefinitionGraph) is a specialization of DefinitionGraph. Figure 14 presents the definition and restriction graphs of CTDefinitionGraph. The definition graph states that a CTDefinitionGraph has at least one concept with a question mark and the restriction graph states that a CTDefinitionGraph may not have two distinct concepts with question marks.

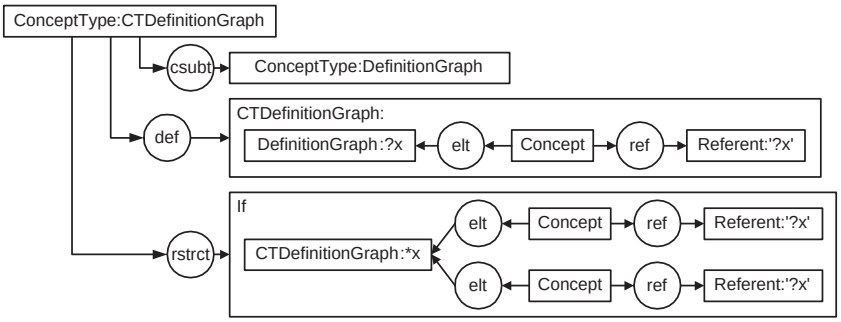

Fig. 14. The definition of a class (type) for Concept Type Definition Graph.

As CTDefinitionGraph above, the CTSyntaxGraph concept type may be defined as a concept type. A Concept Type Syntax Graph (CTSyntaxGraph) is a special-

ization of DefinitionGraph. Figure 15 presents its definition and restriction graphs. The definition graph states that a CTSyntaxGraph has at least one concept and the restriction graph states that a CTSyntaxGraph may not have two distinct concepts.

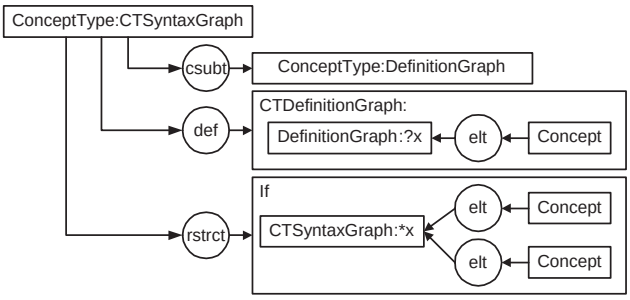

Fig. 15. The definition of a class (type) for Concept Type Syntax Graph.

In this section we demonstrated how the definition primitives of Section 3 and the mapping operator of Section 4 can be used to describe objects of any level by definitions at their metalevel. In the next section we will show how we can complete specifications by adding metarules.

6 Metarules

This section presents metarules we introduced in earlier sections.

The first metarule, as illustrated in Figure 16, states that the definition graph of a concept type has a concept whose referent is `?x`, and whose concept type is in accordance with the concept type which is the destination concept of the csubt relation.

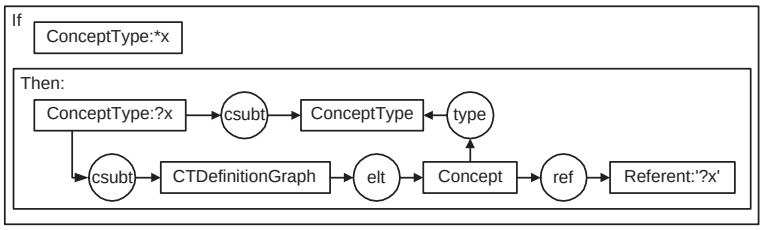

Fig. 16. Concept Type Definition Graph Composition Rule.

The second metarule presented here, illustrates the composition rule for concept type syntax graphs. The metarule (see Figure 17) states that the concept type of the concept that appears in the syntax graph is the concept type itself.

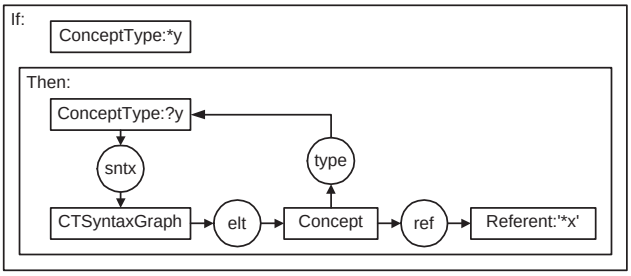

Fig. 17. Concept Type Syntax Graph Composition Rule.

In section 3.2 we argued that a metalevel rule can be expressed to verify that the definition of a relation type r conforms to the relation type of its supertype r' (according to the rsubt relation). Figure 18 presents this metarule.

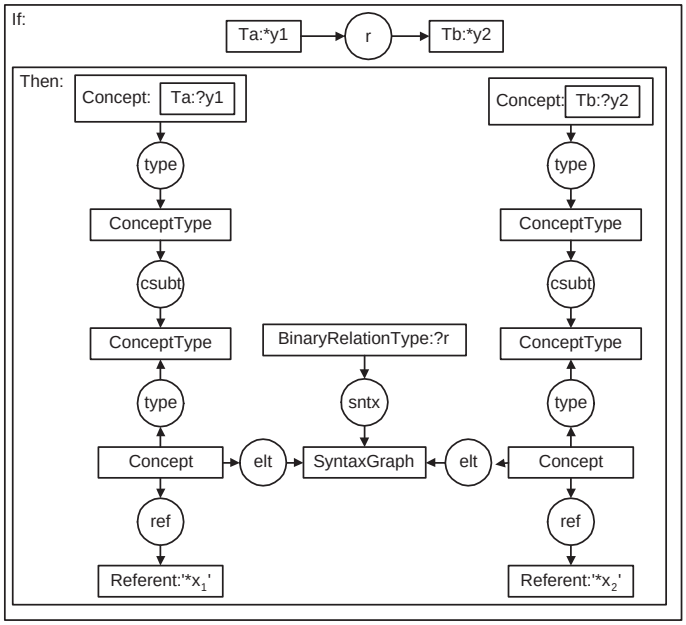

Fig. 18. Signature Compliance Rule.

If two concepts [Ta:*y1] and [Tb:*y2] are linked by a relation whose relation type is r then the two types Ta and Tb are in csubt relation with the two types of the signature of r as expressed in its syntax graph. Concept [Ta:*y1] identified as the source of the relation has a type that is a specialization of the concept type of the concept (element of the syntax graph) whose referent is $*x_1$. Respectively,

Concept [Tb:*y2] identified as the destination of the relation has a type that is a specialization of the concept type of the concept whose referent is $*x_2$.

These few examples demonstrate that conceptual graphs can easily be used to state restriction or define rule at a metalevel.

7 Conclusion and Future Work

Metamodeling and therefore metamodels are important because they formally define the modeling primitives used in modeling activities. In this paper we introduced basic building blocks in order to use Conceptual Graphs in metamodeling activities.

We have seen a metamodeling approach is important because it allows declarative and formal definition of modeling constructs (Section 3 and 5). It authorizes validation of acquired knowledge through formal definition and metarules(Section 6).

In this paper we demonstrated that CGs are powerful enough to be used as an universal metamodeling language but a lot of work remains to be done to define a complete metamodeling framework based on CGs. In [9] we demonstrated that CGs may be used to model the main model component of KADS [22, 23] (a methodology to develop knowledge-based systems), and more generally, we are currently working on the development of a meta-metalevel where we could, using formal definition of modeling languages, specify a mapping between modeling languages. This will allow integration of information systems even if based on different paradigms.

References

[1] T. Cao and P. Creasy. Fuzzy order-sorted logic programming in conceptual graphs with a sound and complete proof procedure. In *Lecture Notes in Artificial Intelligence #1453*, pages 270–284. Springer-Verlag, 1998.

[2] S. Crawley, S. Davis, J. Indulska, S. McBride, and K. Raymond. Meta information management. In *Formal Methods for Open Object-based Distributed Systems Conference*, Canterbury, UK, July 1997.

[3] S. Crawley, S. Davis, J. Indulska, S. McBride, and K. Raymond. Meta-meta is better-better! In *IFIP WG 6.1 International Working Conference on Distributed Systems*, October 1997.

[4] J. Esch. Temporal intervals. In T. Nage, J. Nagle, L. Gerhloz, and Eklund P., editors, *Conceptual Structures*, pages 363–380. Ellis Horwood, 1992.

[5] J. Esch. Contexts as white box concepts. In G. Mineau, B. Moulin, and J. Sowa, editors, *Proceedings of the 1st International Conference on Conceptual Structures (ICCS'93)*, pages 17–29, Quebec City, Quebec, Canada, August 1993. Springer-Verlag.

[6] J. Esch. Contexts and concepts, abstraction duals. In W. Tepfenhart, J. Dick, and J. Sowa, editors, *Proceedings of the Second International Conference on Conceptual Structures (ICCS'94)*, pages 175–184, College Park, Maryland, USA, August 1994. Springer-Verlag.

[7] J. Esch. Contexts, canons and coreferent types. In W. Tepfenhart, J. Dick, and J. Sowa, editors, *Proceedings of the Second International Conference on Conceptual Structures (ICCS'94)*, pages 185–195, College Park, Maryland, USA, August 1994. Springer-Verlag.

[8] D. Gardiner, B. Tjan, and J. Slagle. Extended conceptual structures notation. In *Proceedings of the 4th Annual Workshop on Conceptual Structures. IJCAI-89*, 1989. Section 3.05.

[9] O. Gerbé. *Un modèle uniforme pour la modélisation et la métamodélisation d'une mémoire d'entreprise*. PhD thesis, Université de Montréal, 2000.

[10] B. Ghosh and V. Wuwongse. Computational situation theory in the conceptual graph language. In *Lecture Notes in Artificial Intelligence #1115*, pages 188–202. Springer-Verlag, 1996.

[11] C. Havens. Enter, the chief knowledge officer. *CIO Canada*, 4(10):36–42, 1996.

[12] P. Kocura. Semantics of attribute relations in conceptual graphs. In G. Ganter, B. Mineau, editor, *Proceedings of 8th International Conference on Conceptual Structures (ICCS2000)*, pages 235–248, Darmstadt, Germany, August 2000. Springer.

[13] G. Mineau. Constraints and goals under the conceptual graph formalism: One way to solve the scg-1 problem. In W. Tepfenhart and W. Cyre, editors, *Proceedings of the 7th International Conference on Conceptual Structures*, pages 334–354, Blackburg, VA, USA, July 1999. Springer.

[14] G. Mineau. The engineering of a cg-based system: Fundamental issues. In B. Ganter and Mineau G., editors, *Proceedings of the 8th International Conference on Conceptual Structures*, pages 140–156, Darmstadt, Germany, August 2000. Springer-Verlag.

[15] G. Mineau. The extensional semantics of the conceptual graph formalism. In B. Ganter and G. Mineau, editors, *Proceedings of the 8th International Conference on Conceptual Structures (ICCS'2000)*, pages 221–234, Darmstadt, Germany, August 2000. Springer.

[16] G. Mineau and O. Gerbé. Contexts: A formal definition of worlds of assertions. In *Proceedings of 5th International Conference on Conceptual Structures (ICCS'97)*, pages 80–94, Seattle, Washington, USA, August 1997.

[17] G. Mineau and R. Missaoui. *Semantic Constraints in Conceptual Graph Systems*. DMR Consulting Group Inc., Montreal, Quebec, Canada, June 1996. Internal Research Report #960611A. 39 pages.

[18] G. Mineau and R. Missaoui. The representation of semantic constraints in conceptual graph systems. In D. Lukose, H. Delugach, M. Keeler, L. Searle, and J. Sowa, editors, *Proceedings of 5th International Conference on Conceptual Structures (ICCS'97)*, pages 138–152. Springer-Verlag, 1997. LNAI #1257.

[19] B. Moulin. The representation of linguistic information in an approach used for modeling temporal knowledge in discourses. In G. Mineau, B Moulin, and J. Sowa, editors, *Proceedings of 1st International Conference on Conceptual Structures (ICCS'93)*, pages 182–204, Quebec City, Quebec, Canada, August 1993. Springer.

[20] B. Moulin and S. Dumas. The temporal structure of a discourse and verb tense determination. In J. Tepfenhart, J. Dick, and J. Sowa, editors, *Proceedings of Fourth International Conference on Conceptual Structures (ICCS'94)*, pages 45–68, College Park, Maryland, USA, August 1994. Springer-Verlag.

[21] H. Pfeiffer and R. Hartley. Additions for set representation and processing to conceptual programming. In *Proceedings of the 5th Annual Workshop on Conceptual Structures. AAAI-90*, 1990. Section A.15.

[22] A. Schreiber, B. Wielenga, H. Akkermans, W. Van de Velde, and A. Anjewierden. CML: The CommonKADS conceptual modelling language. In L. Steels, A. Schreiber, and W. Van de Velde, editors, *Proceedings of the 8th European Knowledge Acquisition Workshop (EKAW'94)*, pages 1–24, Hoegaarden, Belgium, 1994. Springer-Verlag.

[23] G. Schreiber, B. Wielenga, R. de Hoog, H. Akkermans, and W. Van de Velde. CommonKADS: A comprehensive methodology for KBS development. *IEEE Expert*, pages 28–36, December 1994.

[24] J. Sowa. Relating diagrams to logic. In John F. Sowa Guy W. Mineau, Bernard Moulin, editor, *Proceedings of the First International Conference on Conceptual Graphs (ICCS'93)*, volume 699, pages 1–35, Quebec City, Quebec, Canada, August 1993. Springer-Verlag.

[25] J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, 1984.

[26] M. Wermelinger. Conceptual graphs and first-order logic. In G. Ellis, R. Levinson, W. Rich, and J. Sowa, editors, *Proceedings of the Third International Conference on Conceptual Structures (ICCS'95)*, pages 323–337, Santa Cruz, CA, USA, August 1995. Springler-Verlag.