Nested conceptual graphs for information fusion traceability

Claire Laudy[1] and Charlotte Jacobé De Naurois[1]

Thales, Palaiseau, France
{claire.laudy, charlotte.jacobedenaurois}@thalesgroup.com

Abstract. InSyTo is a toolbox of algorithms for information fusion and query relying on the conceptual graphs formalism and subgraph isomorphism search. InSyTo was used in order to develop many applications in different domains. Although the framework was used in several application domain and well received by end-users, they highlighted an urgent need for traceability within the information fusion process. We propose here an improvement of the toolbox, in order to embed traceability feature inside the fusion algorithm. The underlying conceptual graph representation of the information was extended from basic conceptual graph to Nested Typed Graphs. A lineage nested graph is added to each concept of the initial information graph, that contains it's processing history through the several processing steps. The lineage graph contains the information concerning the initial sources of each elementary information item (concept), as well as the fusion operations that were applied on them. The main advantage of this new development is the capacity of having a trustworthy framework aware of the current observed situation, as well as the interpretations that were used to build this situation from elementary observations coming from different sources. In this paper, after presenting the context of our work, we recall of the InSyTo toolbox approach and functionalities. We then define the new information representation and operations that we proposed for a matter of traceability handling. The proposition is illustrated on an application we developed recently.

Keywords: Conceptual graphs · Information fusion · Traceability

1 Introduction

With the aim of developing systems that use semantic information for decision support, we chose to use the InSyTo toolbox that provides functions for soft information fusion and management and was previously developed. InSyTo is based on generic algorithms for semantic graphs fusion and comparison that may be adapted to a specific application domain through the use of an ontology. It was used in many projects in the past, as different as crisis management,marketing content design or oceanographic observation.However, if the use of soft information was very much appreciated by these end-users, they also expressed the need to understand where the synthesized information comes from. They express the need for traceability capability within situation awareness and decision support systems provided. Indeed, traceability is highlighted in different guidelines and recommendations for a trustworthy AI [10] [9].

In this paper, we propose an approach for traceability handling as an extension of InSyTo. InSyTo functions rely on the use of Conceptual Graph formalism for semantic information representation. Our approach to traceability is based on the use of nested conceptual graphs, in order to express, for each elementary component of the information, a lineage graph that expresses the whole 'history" of the information item throughout its evolutions regarding fusion and aggregation processes.

The paper is organised as follows. Section 2 provides the context and related works. It recalls the needs that we faced in previous and current projects regarding traceability and presents the basics of the InSyTo toolbox, emphasizing on the fusion approach that is used, regarding which traceability is an important issue. In section 3, we describe the evolution we have performed in order to embed traceability capacity within the toolbox operations. After presenting the general approach, we define formally our proposition. Section 4 finally discusses and concludes our paper and presents future directions for our work.

2 Context

2.1 Conceptual graphs as formalism for information representation

Our aim is to develop systems for which the interaction with human operators is crucial in order to understand the results of the different processes. These results must be easily understandable, as well as should be the integrated analysis processes that lead to these results, such as information aggregation and fusion.

Therefore, the knowledge representation must easily be understood. Therefore, we propose to rely on the use of conceptual graphs. Conceptual graphs (CG) have been formalized by Mugnier and Chein [3] as finite bipartite labelled graphs where the building units of CG are types. Such graphs were proposed by J. Sowa in [11] as a graphical representation of logic. They allow representing knowledge in a easily readable manner for humans, experts of specific application domain, but non experts of knowledge representation formalism.

Several tools exist in order to create conceptual graphs knowledge bases and interact with them. Among them, Cogui [7] enables one to create a Conceptual graphs knowledge base and offers imports and exports capabilities to different semantic web formats. Cogitant [5] is a C++ platform that provides capacities to represent the different elements of the conceptual graphs model, as well as functions to reason over and manipulate the graphs. InSyTo is a JAVA toolbox of algorithms for information fusion and query relying on the conceptual graphs formalism and sub-graph isomorphism search proposed in [2]. It was further improved with uncertainty management capabilities [4] which enable to manage some of the imperfection that may be found on reports provided by humans. For the information aggregation and fusion part of our work, we chose to use the functions provided by InSyTo for the possibility to add domain knowledge in order to tune the functions. We describe the toolbox hereafter.

2.2 InSyTo : a toolbox for information fusion

InSyTo is a toolbox that contains several core functions. These functions can be combined in order to provide advanced semantic information management functions. InSyTo core functions are are depicted in Fig. 1 and described hereafter.

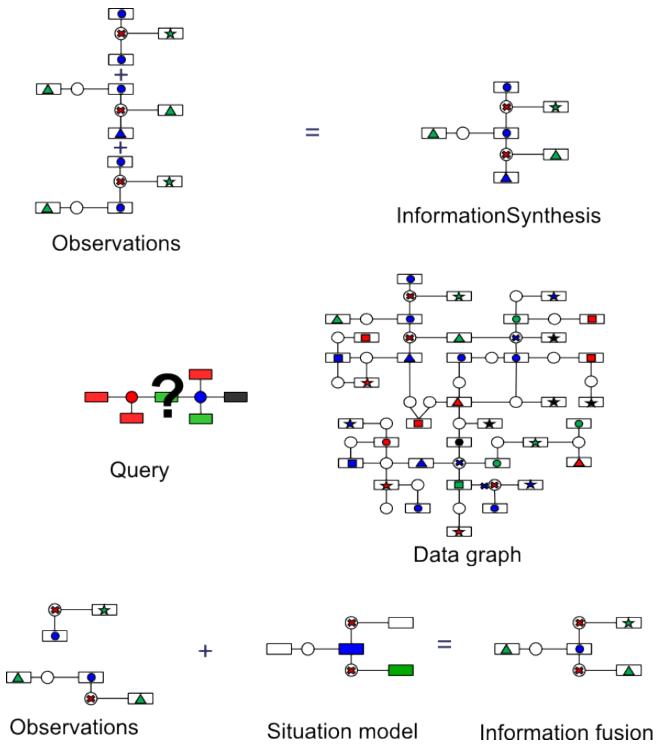

Fig. 1. InSyTo core functions

Information Synthesis Information synthesis enables one to collect and organize information about a specific subject. Through information synthesis, all the gathered information items are organized into a network. The redundant part of the information items are detected and eliminated.

The fusion strategies are used within information synthesis, in order to enable the fusion of information items that are slightly different but describe the same situation of the real life. These discrepancies may appear when different sources of information with potentially different levels of precision for instance, are used to draw a picture of an on-going situation.

Information Query All the instances of information corresponding to a specified graph pattern may be found within a network of information, through the **information query** function.

The specialization relationship between the query and the data graphs imply that the structure of the query graph must be entirely found in the data graph. The query function relies on the search for injective homomorphism between the query graph and the data graph.

Information Fusion When a model of a situation of interest (e.g. an activity involving a specific person at a specific date) is available, one may want to monitor the situation and trigger further processes if an instance of such a situation is happening. Therefore, different observations, potentially coming from different sources, are filtered out in order to keep observations of interest only. They are then assembled through **information fusion** in order to provide a representation of the ongoing situation of interest, as precise as possible.

The model of situation is, within information fusion, more generic than the observation graphs. Further more, fusion strategies may be used, as for the Information Synthesis function. The use of the model constraints the structure of the fused observation.

To implement these core function, InSyTo encompasses a generic graph-based fusion algorithm made of two interrelated components (see Fig 2). The first component is a generic sub-graph matching algorithm, which itself relies on the use of fusion strategies. The graph matching component takes care of the overall structures of the initial and fused observations. It is in charge of the structural consistency of the fused information, regarding the structures of the initial observations, within the fusion process.

The fusion strategy part is made of similarity, compatibility and functions over elements of the graphs to be fused. They enable the customization of the generic fusion algorithm according to the context in which it is used.

2.3 The need for traceability in information fusion

InSyTo is used in order to develop high level information managing functions such as alarm raising, event detection, inconsistency detection in reports. It was deployed on several projects ranging from crisis management to investigation and oceanography. In many of these application, if the management of information coming from soft information sources (social media, police reports...) was a new and interesting topic, the need for traceability of the information was raised.

Indeed, when multiple fusions occurs on graphs, it could be necessary to know where and how this information was constructed, to find and/or follow the history of the fusion process and to measure the ability of the fusion system to provide an accurate and unbroken historical record of its inputs and the chain

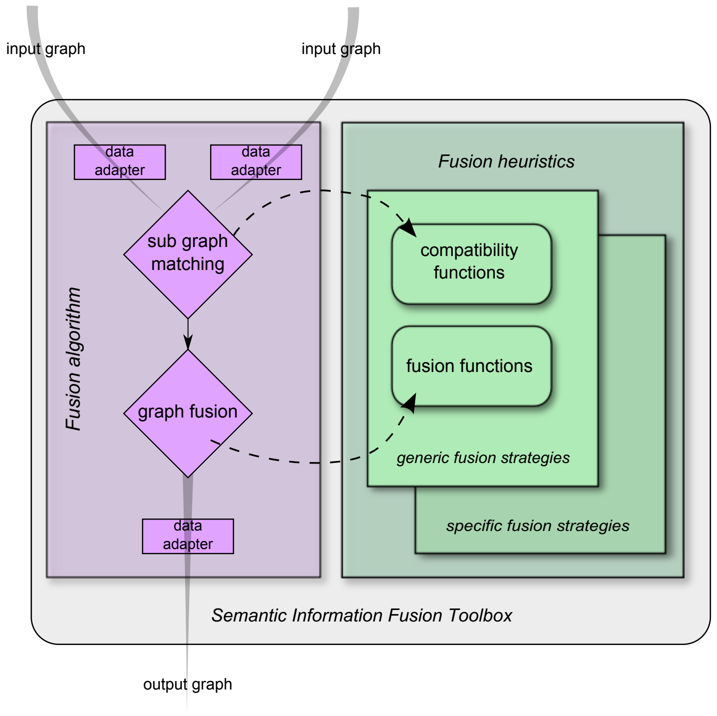

Fig. 2. InSyTo algorithm

of operations that led to its conclusions. Keeping trace of a all fusion processes in a way that is verifiable and reproducible is necessary for different use-cases.

This feature is called **traceability**. Traceability is known to be applicable to measurement, supply chain, software development, healthcare and security also. On software development, the traceability consists of following relations between software artifacts and requirements during the development life cycle [6]. In this context, ontology is sometimes used to automatically keep a trace of requirements [1], or artifacts between source code and documentation [13].

To the best of our knowledge, no work was proposed for handling traceability with semantic information and information sources in the context of semantic information fusion.

3 Tracing information Fusion with InSyTo

3.1 General approach

When keeping awareness of an on-going situation, all the reports and testimonies about that situation are consolidated within what we call a *situation graph*. This situation graph is the synthetic summary of all the information one has, provided by all the available information sources. Furthermore, information sources each provide what we call initial information items. These items are aggregated and fused together into the situation graph.

So, to keep a trace of the initial information items that resulted in a situation graph, we need to track the successive fusion operations that were applied to each one of its concepts. For tracking all fusions of concepts, we propose to add a *lineage graph* inside each concepts node.

At the beginning, before the fusion process, the lineage graph of a specific concept node contains specific information about the source of this information.

3.2 Background : Simple, nested and typed conceptual graphs

Conceptual graphs (CGs) [3] are a family of formalisms of knowledge representation, made of ontological and factual knowledge. They are bipartite graphs defined over ontological knowledge stored in the vocabulary.

The ontological part of a CG is a *vocabulary*, defined as a 5-tuple $V = (T_C, T_R, \sigma, I, \tau)$. T_C and T_R that respectively correspond to concept and relation types are two partially ordered disjoint finite sets, where ordering corresponds to generalisation. An example of T_C, used in the illustration below is given in Fig. 3.2. It contains a greatest element $\top$. T_R is partitioned into subsets $T_R^1 \dots T_R^k$, $1 \dots k$ $(k \geq 1)$ respectively, meaning that each relation type has an associated fixed arity. σ is a mapping associating a signature to each relation. I is a set of individual markers. τ is a mapping from I to T_C.

The conceptual graphs themselves represent facts. In our work, they represent the situations that are observed. A CG is a 4-tuple *G = (C,R,E,label)*. G is a bipartite labeled multi-graph as illustrated on Fig. 4. A CG is made of concept and relation nodes. On Fig. 4, the rectangular boxes represent concept nodes and the ovals represent relation nodes. C and R correspond to concept and relation nodes, where elements of C are pairs from $T_C \times I$ and elements of R are elements of T_C. E contains all the edges connecting elements of C and R and *label* is a labelling function.

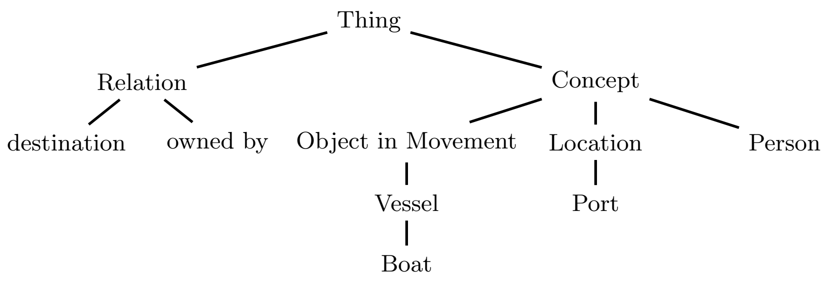

Fig. 3. A simplified vocabulary example

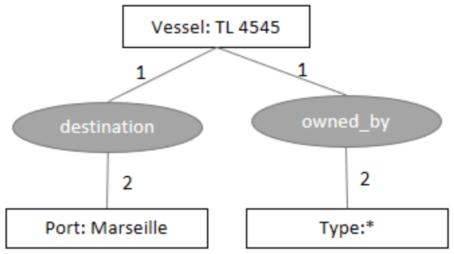

Fig. 4. An example of conceptual graph $G1$. The rectangular boxes represent concept nodes and the ovals represent relation nodes.

Concept nodes are made of a conceptual type defined in T_C and an individual marker. The term **concept** is used to refer to a concept node. The concepts represent the "things" or entities that exist. A concept is labeled with two components: the conceptual type and the individual marker.

The **conceptual type** defines the category to which the entity belongs. For instance, in Fig. 4 the concept `[Port: :Marseille]` is an instance of the category `Port`, i.e., its conceptual type is `Port`.

The **individual marker** relates a concept to a specific object of the world. The object represented by `[Port: :Marseille]` has the name (or value) `Marseille`. The individual markers may also be undefined. An undefined or generic individual marker is either blank or noted with a star `*`, if the individual object referred to is unknown.

The term **relation** is used to refer to a relation node. The relation nodes of a conceptual graph indicate the relations that hold between the different entities of the situation that is represented. Each relation node is labeled with a relation type that points out the kind of relation that is represented.

Nested Conceptual graphs are an extension of basic conceptual graphs.They are used in order to provide different levels of knowledge related to concepts of a graph. While concepts and relations linked to a concept provide external contextual information about the concept, internal information about the concept may be provided as graph, nested inside the concept. Chein and Mugnier provide the didactic example in Fig. 5, where a drawing, made by the boy Paul sets on a table (external contextual knowledge). Furthermore, this drawing represents a green train (internal information).

In addition to their conceptual types and individual markers, each concept c of a nested conceptual graph encompasses a third element called *description* of c and denoted $Descr(c)$ in [3]. $Descr(c)$ iteself contains either a nested conceptual graph that describes the contents of the concept c or the value $**$ meaning that no further nested description is available.

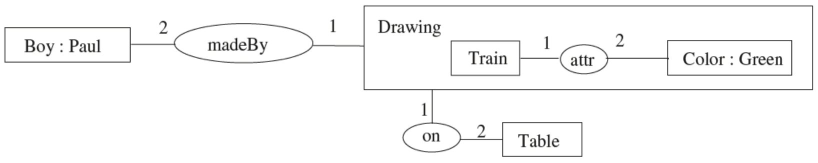

Fig. 5. A nested conceptual graph.

Nested Typed Graphs are used when the universe of discourse is broken into independent parts. In our study, the situation graph and its set of lineage nested graphs define different parts of knowledge that may not be mixed. Thus nested typed graphs appear to be naturally well suited in order to represent the tracked situation graphs, broken into the situation itself and the historical knowledge of the fusion operations over the situation concepts.

3.3 Nested typed graphs as concepts' lineage

As explained in the general approach, the traceability of each concept of a fused situation graph is a property of this concept containing internal information about the its state and its so called *history* through the fusion operations. Thus, we add a *nested lineage graph* to each concept node of an information graph. For each concept c of a situation graph, this lineage graph is added to the concept, additionally to the type and marker that the basic conceptual graph already contains, as the value of $Descr(c)$.

The lineage graph of a concept is a nested typed graph, defined on a specific vocabulary and with a limited number of relations that we define hereafter. The situation graph is defined on a different vocabulary than the lineage graphs. The two sets of graphs, situation and lineage will never be mixed during the fusion of 2 situation graphs. However, the fusion of two situation graphs modifies the set of lineage graphs as defined hereafter. To emphasize on this distinction, we use typed graphs for situation and lineages graphs, each having a different type from the set $Situation, Lineage$. Figure 7 presents the tree of graph types for our tracked situation graphs.

At the beginning, before the fusion process, the lineage graph of a specific concept node contains specific information about the source of this information. Figure 6 depicts a example of lineage graphs before a fusion operation has occur.

In order to ease the understanding, in the following definitions we refer to elements of a situation graph such as concepts and relations using the adjective *situation* (*situation concept* and *situation relation* for instance), while we use the adjective *lineage* for the elements of the lineage graph.

Definition 1 (Tracked situation Graph). *A* tracked situation graph S *is a nested typed basic graph. The graph* S *is of type* Situation*.*

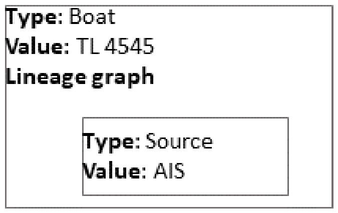

Fig. 6. Example of the definition a concept with type: Boat, Value: TL4545 and lineage graph. The Lineage graph is composed of type: Source, Value: AIS, that is to say the $[Boat : TL4545]$ cames from AIS source.

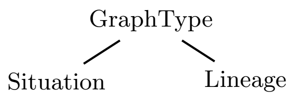

Fig. 7. Types of graphs used within InSyTo

Definition 2 (Tracked situation concept). *Each concept node of a tracked situation graph is called a* tracked situation concept*. The field* description *of a* tracked situation concept *node* c *contains the lineage graph of c, denoted* $Lineage(c)$*.*

Definition 3 (Lineage graph). *The* lineage graph *of a tracked situation concept* c *is a typed basic conceptual graph denoted* $Lineage(c)$*.* $Lineage(c)$ *is of type* Lineage*. It is defined on a vocabulary* $T_{lineage} = (C_L, R_L, Lineage, I)$*, where* C_L *and* R_L *are respectively the partially ordered set of concepts and relations of the lineage graph defined defined hereafter.*

It is to note that as the lineage graph of a concept c describes the evolution of c through the sequence of fusion operations that were applied on different instances of this concept, the c and its lineage graph share multiple co-reference links. However, as each level of the lineage graph describes a different context of observation of c (i.e. the successive states of c and its successive values), the co-referent concepts should never be merged.

Definition 4 (Lineage concept types). *Let* $Lineage(c)$ *be the lineage graph of the concept c and let* $type(c)$ *be the type of the concept c.* $Lineage(c)$ *is defined on a vocabulary* $T_{lineage} = (C_L, R_L, Lineage, I)$ C_L *is the partially ordered set of types composed of the union of the ancestor and descendant of c,* $type(c)$*, the uncomparable types* Source*,* FusionStrategy*,* FusionFunction *and* SimilarityMeasure *and their descendants. It is to note that the specific type* FusionStrategy *is a known of the direct subtype of* Source

$$\begin{aligned} C_L = & type(c) \cup ancestor(type(c)) \cup descendant(type(c)) \\ & \cup Source \\ & \cup FusionStrategy \cup descendant((FusionStrategy)) \\ & \cup FusionFunction \cup descendant((FusionFunction)) \\ & \cup SimilarityMeasure \cup descendant(SimilarityMeasure)) \end{aligned} \tag{1}$$

The root part of the tree of conceptual types C_L is depicted on figure 3.3. The *FusionStrategy*, *FusionFunction* and *SimilarityMeasure* types may have additional sub-types, according to the fusion strategies defined and used within the InSyTo specific domain application. For a matter of readability, we noted *TrackedConceptType* the sub tree composed of the $type(c) \cup ancestor(type(c)) \cup descendant(type(c))$.

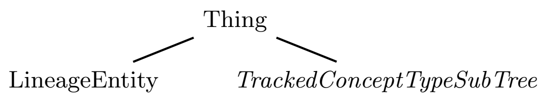

Fig. 8. set of concept types for the lineage graphs

Definition 5 (Lineage relations). *The set of lineage relation types R_L is unordered and the set of lineage relations signatures is composed as follows.*

- *compatibility_function : FusionStrategy* $\mapsto$ *SimilarityMeasure * Threshold*
- *fusion_function : FusionStrategy* $\mapsto$ *FusionFunction*
- *produced_by : FusionStrategy* $\mapsto$ *TrackedConceptTypeeSubTree * TrackedConceptTypeeSubTree*

3.4 Information fusion

Fusion strategies are heuristics that are part of the merge operation of concepts between conceptual graphs. They are rules encoding domain knowledge and fusion heuristics that are used to compute the fused value of two different but compatible concept nodes. On the one hand, the fusion strategies extend the notion of compatibility that is used in the maximal join operation. According to some fusion strategy, two entities with two different values may be compatible and thus fusable. On the other hand, the strategies encompass functions that give the result of the fusion of two compatible values.

Fusion strategies The fusion functions available in the toolbox are are expressed as the composition of two functions:

Let E be the set of concept nodes defined on a support $\mathcal{S}$. Let G_1 and G_2 be two conceptual graphs defined on $\mathcal{S}$. A fusion strategy $strategy_{fusion}$ is defined as follows :

$$strategy_{fusion} = f_{fusion} \circ f_{comp} : E \times E \rightarrow E \cup \{E \times E\}$$

where $f_{comp} : E \times E \rightarrow \{true, false\} \times E \times E$ is a function testing the compatibility of two concept nodes,
and f_{fusion} is a fusion function upon the concepts nodes of the graphs. the extended version of f_{fusion}, taking into account the nested lineage graph, will be formally defined in the next section.

The compatibility function is defined the similarity between the values (markers) of two concept nodes. This similarity is defined by domain experts, given the requirements of the application. The similarity measure is compared to a threshold defined by domain experts, thus the compatibility function f_{comp} is then defined as follows :

$$f_{comp}(c_1, c_2) = sim(c_1, c_2) \geq threshold_{sim}$$

The fusion strategies applied on two concept nodes result either in a fused concept node if the initial nodes are compatible, or in the initial nodes themselves if they are incompatible.

3.5 Tracked Fusion of situation graphs

As fusion strategies my not be a commutative operation, according to the specific fusion functions used, there is a need for traceability when applying several fusion operations on several situation graphs into a single situation graph. We define hereafter the tracked fusion operation over two situation graphs. The fusion strategies are not impacted by the tracking of fusion history into the lineage graph, however, their use must be memorized in the lineage graph.

f_{fusion} is a higher level function, taking into parameter three fusion functions, each one applying on one of the element of the concept node :

- $fusion_{type}(c_1, c_c2) = most\_general\_subtype(type(c_1), type(c_c2)) = type(c)$
- $fusion_{marker}(c_1, c_2 = fusion(c_1, c_2))$ where $fusion : \{true, false\} \times E \times E \rightarrow E \cup \{E \times E\}$ is a fusion function upon the concepts nodes of the graphs that is application dependant.
- $fusion_{lineage}(c_1, c_2)$ is the fusion operator on lineages defined hereafter.

Definition 6 (Lineage graphs fusion operation). *Let G_1 and G_2 be two situation graphs defined on $V = (T_C, T_R, \sigma, I, \tau)$.*

Let $G_1 = (C_1, R_1, E_1, label_1)$ and $G_2 = (C_2, R_2, E_2, label_2)$.

Let c_1 and c_2 be two concepts defined of respectively G_1 and G_2. Let L_1 and L_2 be the lineage graphs of respectively c_1 and c_2.

The fusion operator over lineage graphs is defined as follows:

$$fusion_{lineage} : E_1 \times E_2 \rightarrow E_L,$$

with E_L the set of concepts of the fused lineage graph L.

The result of the fusion of the two lineage graphs is as follows:

$$\begin{aligned} fusion_{lineage}(c_1, c_2) = & \quad L \\ = & \quad produced\_by(FusionStrategy(f), c_1, c_2) \\ & \wedge compatibility\_function(FusionStrategy(f), \\ & SimilarityMeasure(sim(c_1, c_2)), \\ & Threshold(threshold_{sim})) \\ & \wedge fusion\_function(FusionStrategy(f), \\ & FusionFunction(fusion_{marker}(c_1, c_2))) \end{aligned}$$

L is the nested typed graph depicted on the green part of theFig.9

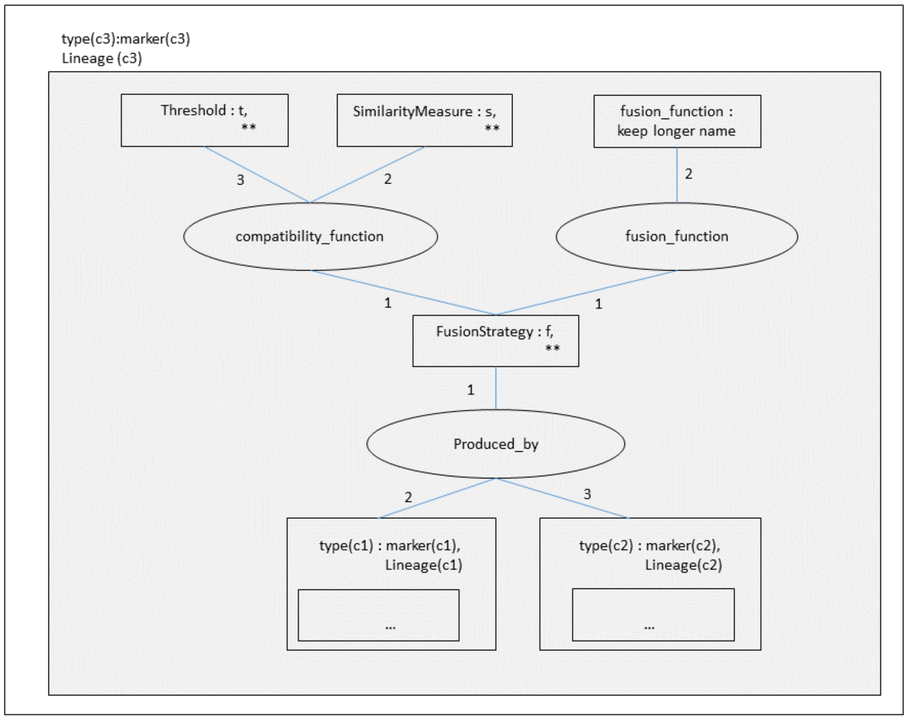

Fig. 9. Concept c_3 (fusion of (c_1 and c_2) and its lineage graph resulting from the fusion of the two concepts c_1 and c_2 in the green part.

4 Discussion and conclusion

We proposed an improvement to the InSyTo toolbox which enables traceability capacity as the the use of nested graph with a lineage graph in order to embed traceability feature inside the fusion algorithm. A lineage nested graph is added to each concept of the initial information graph, that contains it's processing history through the several processing steps. The lineage graph contains the information concerning the initial sources of each elementary information item (concept), as well as the fusion operations that were applied on them.

This improvement enables the end-users to understand where the synthetic information in the situation graph comes from. Thanks to the lineage graphs of each concepts of the situation graph, one may rebuild the whole aggregation and fusion process, even in cases where the fusion strategies are not commutative. We developed a first use case and were able to visualize the operations achieved on information items. Adding traceability to the applications developed with the framework improves the trust end-users have on the system through the availability of explainable elements over the underlying and aggregation fusion processes.

However, if we provided the tools to handle traceability, there still remains a need for easy visualisation and analysis of the trace information (for example [12]). With that purpose in mind, the use of nested conceptual graphs has the great advantage of enabling one to use the same analysis and manipulation functions as for the analysis of the situation graph. Graph based and conceptual graphs based algorithms can be used in order to provide end-users with analysis capacities over the trace graphs. On can achieve a search for specific sources of information, or statistics of the fusion operations used and the global level of similarity of individual information items, for instance, using well known, tried and tested graph algorithms.

Furthermore, representing the concept's lineage as an nested graph has an other advantage, regarding the easiness of manipulation of information. Indeed, even if the traceability information and the knowledge are integrated into a single nested graph, they can easily be isolated in two different smaller conceptual graphs $[situation, lineage]$. This would be achieved by somehow "cutting" the nested situation graph at the first level of lineage, that is to say cutting the lineages of the situation concepts and storing them in a separated graph data base.

Finally, if we chose to use a reduced vocabulary in order to express the trace information in the lineage graphs, our approach can be enriched using other ontology (for example [8]). One can represent the lineage graph with another vocabulary and relations if necessary. Identically, if one wants to improve the toolbox with other operations, their application to a concept node can also be represented in the *lineage*.

5 Acknowledgment

This work was partially performed under the 883347 H2020 project EFFECTOR, which has received funding from the European Union's Horizon 2020 Program. This paper reflects only the authors' view, and the European Commission is not liable to any use that may be made of the information contained therein.

References

1. Assawamekin, N., Sunetnanta, T., Pluempitiwiriyawej, C.: Ontology-based multi-perspective requirements traceability framework. Knowledge and Information Systems **25**(3), 493–522 (2010)
2. Author: Using maximal join for information fusion. In: Graph Structures for Knowledge Representation and Reasoning (GKR 2009) collocated with the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI-09) (2009)
3. Chein, M., Mugnier, M.L.: Graph-based Knowledge Representation: Computational Foundations of Conceptual Graphs. Springer (2008)
4. Fossier, S., Laudy, C., Pichon, F.: Managing uncertainty in conceptual graph-based soft information fusion. In: Proceedings of the 16th International Conference on Information Fusion. pp. 930–937. IEEE (2013)
5. Genest, D., Salvat, E.: A platform allowing typed nested graphs: How cogito became cogitant. In: Mugnier, M.L., Chein, M. (eds.) Conceptual Structures: Theory, Tools and Applications. pp. 154–161. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)
6. Gotel, O.C., Finkelstein, C.: An analysis of the requirements traceability problem. In: Proceedings of IEEE International Conference on Requirements Engineering. pp. 94–101. IEEE (1994)
7. Hamdan, W., Khazem, R., Rebdawi, G., Croitoru, M., Gutierrez, A., Buche, P.: On ontological expressivity and modelling argumentation schemes using cogui. In: Bramer, M., Petridis, M. (eds.) Research and Development in Intelligent Systems XXXI. pp. 5–18. Springer International Publishing, Cham (2014)
8. Lebo, T., Sahoo, S., McGuinness, D., Belhajjame, K., Cheney, J., Corsar, D., Garijo, D., Soiland-Reyes, S., Zednik, S., Zhao, J.: Prov-o: The prov ontology (2013)
9. Sharma, S., Henderson, J., Ghosh, J.: Certifai: Counterfactual explanations for robustness, transparency, interpretability, and fairness of artificial intelligence models. arXiv preprint arXiv:1905.07857 (2019)
10. Smuha, N.A.: The eu approach to ethics guidelines for trustworthy artificial intelligence. Computer Law Review International **20**(4), 97–106 (2019)
11. Sowa, J.F.: Conceptual Structures. Information Processing in Mind and Machine. Addison-Wesley, Reading, MA (1984)
12. Ware, C.: Information visualization: perception for design. Morgan Kaufmann (2019)
13. Zhang, Y., Witte, R., Rilling, J., Haarslev, V.: An ontology-based approach for traceability recovery. In: 3rd International Workshop on Metamodels, Schemas, Grammars, and Ontologies for Reverse Engineering (ATEM 2006), Genoa. pp. 36–43 (2006)