Visual CP Representation of Knowledge

Heather D. Pfeiffer and Roger T. Hartley

Department of Computer Science
New Mexico State University
Las Cruces, NM 88003-8001, USA
email: hdp@cs.nmsu.edu and rth@cs.nmsu.edu

Abstract

Knowledge can be expressed in two forms: declarative and procedural. Declarative knowledge describes a set of definitions about the world, while procedural knowledge describes the temporal, spatial and constraint aspects applied within these definitions. Expressing declarative knowledge textually is in some ways difficult to follow, whereas it is virtually impossible to capture all aspects of procedural knowledge in this manner. A visual representation of both forms of knowledge is both more desirable and actually more descriptive. Through a visual representation that uses graphs, CP, the two forms can give a balanced view of knowledge while capturing the meaning of both.

Keywords: Diagrammatic Languages, Data-Flow Languages, Graph Manipulations, Conceptual Structures and Knowledge Representation

1. Introduction

According to the dictionary, knowledge is ``something learned and kept in the mind; the act of understanding." However, to process knowledge with a computer, there must be some way to represent knowledge to the computer. This problem of describing knowledge is known as *knowledge representation* where representation consists of a set of syntactic and semantic rules describing the understanding of a problem or a problem domain. However, there are two types of knowledge that humans deal with every day, 1) knowledge that defines an idea or concepts and their relationships, and 2) knowledge that gives understanding to time, space, or constraints in connection to these definitions. The first type of knowledge is known as *declarative knowledge*, describing a set of definitions about the world. The second type of knowledge is *procedural knowledge*, describing the temporal, spatial, and constraint aspects for the above definitions. Given that there is a duality between these two types of knowledge [16], one type is inadequate without the other.

Through out history, language has been used to describe knowledge and conceptual relationships. In many instances it is easier to describe in words definitions of concepts and their relationships, i.e. the cat sat on the mat. In this simple example, both types of knowledge are being used; 1) cats related to mats, and 2) spatially, the cat is on the mat. Sometimes, written language is not an easy tool to use to describe an idea, i. e. A rat sat on the mat before a cat sat on the mat. There can be two interpretations of this idea: 1) the rat is sitting in front of the cat on the mat at the same time, or 2) the rat sat on the mat prior to the time the cat sat on the mat. Here a picture or a time diagram (see Figure 1), can display the correct interpretation. Again, both types of knowledge are being used, 1) cats related to mats; rats related to mats, and 2) spatially, the cat on the mat and the rat on the mat; temporally, the rat on the mat before the cat on the mat.

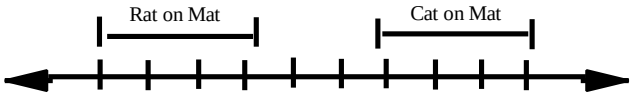

Figure 1: Time Chart

Since it is easier to describe time in pictures as opposed to words, one does not always understand the nature of the time element for a conceptual relationship or a set of conceptual relationships. This sometimes creates a problem when trying to use pure language over a diagram or picture.

When one desires to describe knowledge to the computer, the knowledge representation used must describe and store knowledge for both types. For many knowledge representations, this creates two forms of abstractions that are not necessarily compatible. This causes an imbalance within the representation. We believe that our visual language CP can alleviate this imbalance.

2. Perceived Imbalance

As human beings, we communicate mostly by using written and spoken language, but we live in a visual world. Vision is the most refined of our human senses [5]. When we depict information in our world through the use of pictures or diagrams, we can make breakthroughs in understanding. For example, the mystery of DNA structure was unlocked when the double helix was depicted as a possible visual solution. The visual space relationship is far easier to perceive then time relationship between objects. Writing of language is even spatial in the formation of letters, so language can be seen as having a visual element.

Many knowledge representations are based on a linear representation of thought. Even thinking is linear in time, and best supported in a visual manner. It is natural to design the knowledge representation for declarative knowledge and spatial knowledge first, and then add temporal and other procedural knowledge later. Knowledge representation has been like this from the early days of Aristotle to the modern day. Whether one uses logic, frames, semantic networks or rules, they employ text and/or pictures to capture concepts and their relationships. Therefore, they give a representation for declarative knowledge and spatial procedural knowledge, but try to add in the rest of procedural knowledge later. The absence of a temporal component is obvious. For instance, logic as a knowledge representation represents statements that can be true or false, but does not address the interval of time over which the relationship holds. Therefore, the temporal component is an add-on, and research has been performed in how to put time into the picture [[4] [8]]. All the other representational forms have made similar attempts to add the other procedural components, but they are always added after the initial representational structure has been developed.

Looking closely at the two forms of representation for the two types of knowledge, declarative representation is a set of declarations of conceptual objects and their relationships. The procedural representation allows these declarative representations to be used in action form. It defines how to represent and handle processes involving the concepts and their relationships. With procedural representation, one can reason about the declarative knowledge in order to come to some understanding of it.

The imbalance between the two forms comes because they are not defined and operated on as a unit. As discussed above, the declarative representation and possibly spatial representation is defined first and then the rest of the procedural representation is an add-on. In fact, procedural representations commonly are a set of ways to change the original representation rather than being a representation of the actions themselves. Logic is stuck with *logical inference*, a set of syntactic rules that map onto commonly accepted natural inference results.

Inference rules are permissive, which means that they can be applied whenever one chooses. However, they are inadequate for capturing the intricacies of human thought because they have no rule for processing objects.

The generalization of inference into a rule set for a rule-based system or an expert system also allows the rules to be permissive. In a rule-based system, a rule does not change the logical form of a representation, but allows a rule to change the state of an object (see Section 3) and its relationships. The sequencing of the application of the rules in the rule set can be mapped onto a sequence of actions (see Section 3), but again the temporal component is applied out of the behavior of the rule engine (so-called control knowledge [3]), and is not represented explicitly. This represents the temporal component of the procedural knowledge, but as a change to the rule engine and not as an actual part of the overall representation.

So the imbalance between representing declarative and procedural knowledge comes not only in the amount of effort it takes to define the knowledge representation, but also in the ability of the given knowledge representation to define procedural knowledge.

3. Ontology

One approach to the whole issue of declarative versus procedural is to do some basic *ontology*, what is knowable in the world. In earlier papers [[6] [13]], we make such attempts, within the framework of conceptual graph theory (see Section 4.1). The results were encouraging and had promise. In summary, the known world consists of objects and actions. The objects are conceptual in nature, i.e. a book or a man; the actions are acts between the objects or sometimes relationships, i.e. the man gave the book to another man.

Case relations link objects to actions, i.e. man and giving; spatial relationships exist between objects, i.e. man and book; and temporal relationships exist between actions, i.e. at one moment in time one man has the book, and in an other moment the another man has the book. This can be summarized in the ``canonical square'' (see Figure 2). The canonical square is described such that causal relationships (CR) are all forms of links between two actions and two objects; spatial relationships (SR) between two objects and temporal relationships (TR) are between two actions. Objects can be viewed as declarative while actions can be viewed as procedural.

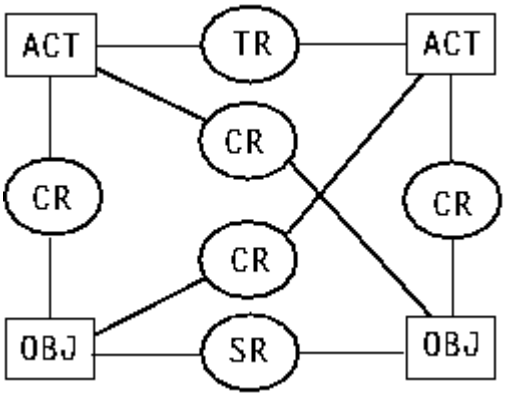

Figure 2: The Canonical Square

As can be seen, the duality of declarative and procedural knowledge is preserved in the symmetry of the ontology. This ontology is also visual. However, this ontology uses only the elements of conceptual graph theory as defined by Sowa [17] which are declarative, concepts and relations; it does not use the actors which can be seen as procedural. It also does not use the extensions added in to the CP language as described in section 4.2 to incorporate procedural knowledge. Temporal relationships appear as just another declarative relationship. This proposes the question of how procedural knowledge will be processed. However, it should be noted, the inadequacy is in the notation of the ontology, not in the ontology itself. In Section 6, we will address this problem while preserving the ontology.

4. Conceptual Programming, CP, Environment

The basic knowledge representation for the Conceptual Programming, CP, environment is semantic networks. As reported earlier [[12] [13]], CP is based upon a graphical methodology of visualization within a semantic network derived from John Sowa's conceptual graph theory [17].

4.1. Conceptual Structures

Conceptual structures, *CS*, as defined in Sowa's book [17], expresses declarative knowledge by implementing it as a connected multilabeled bipartite oriented graph. There exist a mapping from each of these graphs, *conceptual graphs*, to formulae in first-order logic. Each graph uses concept, relation, and actor type nodes, and links the total context together through the edges that connect them. Each label in a concept node, displayed as a box, consists of two fields, the *type* field and the *referent* field. The type field is an element of the set of concepts defined in a type lattice (see [13] for details). The referent field contains the individual specialization (if any) for the type field. Each label in a relation node, displayed as an elliptical circle, consists of a single *relation* field. This relation field depicts the relationship between the adjoining concept nodes within the conceptual structure (see Figure 3).

Sowa has shown how unknown objects (nodes with no individual field) can be computed by firing an *actor* node that corresponds to a function in standard logic. Actor nodes of this kind are diamond-shaped boxes connected to concept nodes with dashed lines. However, even though these actors are procedural in nature, they are still declarative knowledge within the graph.

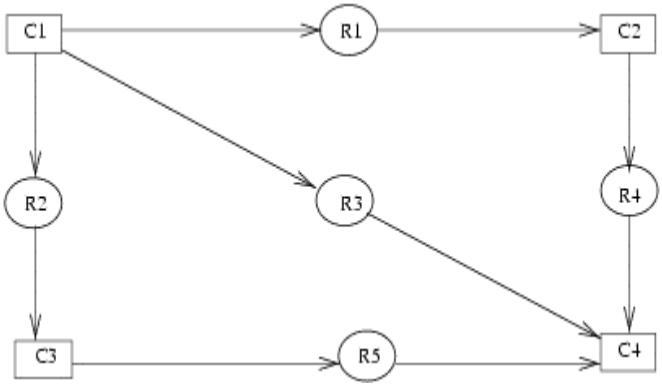

Figure 3: Basic CG Graph

4.2. CP

In CP all knowledge is represented by graphs and operations (mappings) performed on those graphs.

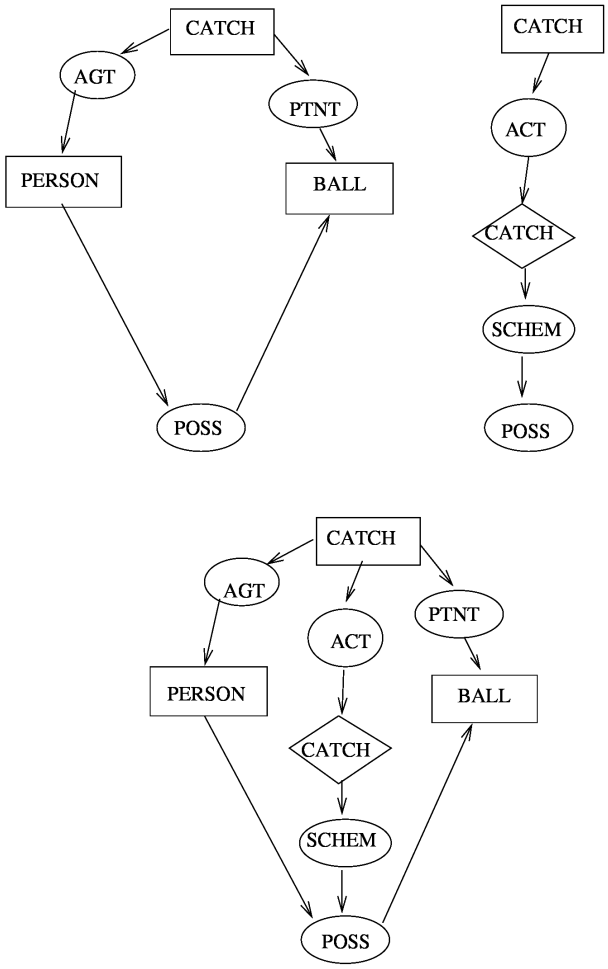

Figure 4: Basic CP graph, actor overlay, complete CP procedural graph

Unlike Sowa's conceptual graphs that only express declarative knowledge using concept, relation, and actor nodes, and link the total context, together through the edges that connect them, CP graphs can express not only declarative, but also procedural knowledge. CP introduces a mechanism for expressing procedural knowledge through extended features to actors, both syntactically and semantically, where they perform more like “functional relations”[[12][13]]. The procedural knowledge is represented as *overlays*, just like overlays on a slide, to the set of definitions that make up the declarative knowledge for a problem domain. These overlays allow the processing of time, space and other constraints. This additional level of knowledge allows for both feasibility-runtime domain support and spatial-temporal domain support. Within the feasibility-runtime domain, heuristics and physical constraints are introduced into the conceptual structures framework; where as, within the spatial-temporal domain an ontology for objects, events, states and processes is provided within the conceptual structures framework. The CP environment incorporates into the graph operations, join and project, internal constraint processing much like meta-rules using this new level. Overlays use actor nodes to overlay functional relations onto the declarative knowledge. In this way, there is a set of semantic rules (performed by a procedure) being represented graphically between objects (see Figure 4)

As seen in the above ontology (section 3), declarative knowledge is captured with static relationships. CP using the overlays treated procedural knowledge in the same way [13]. For example, a set of spatial relationships among objects, called a *spatial snapshot*, will be fixed in time. The time is a single interval in temporal space and will include all objects spatially related within this interval. Also, a set of temporal relationships, called a *temporal snapshot*, will be fixed in space. This fixed space is the union of all the spaces occupied by all the objects involved in the actions during the time interval the actions took place. Even though this does give more of a balance to the knowledge representation, there is still no dynamic representation of the actual actions. Actors are still changing states of other nodes when they are executed instead of conceptually representing the actions. There is no obvious flow of time through the graphs enabling one to answer the question, ``What happens next?". A new idea will be addressed in section 6.

4.3. Related Works

The balance between declarative and procedural knowledge formed part of Peirce's theory of pragmatism, as well as being a constant theme in philosophy over the ages. Rochowiak [15] presented a discussion of this that supports our contention that conceptual structures need a procedural component in order to achieve the unification of the "theoretical and practical dimensions of a concept" [15].

Representation of procedures in conceptual graphs has gone through several stages (apart from our own work), using several programming paradigms. Sowa's himself enhanced his original inclusion of functional actors in the theory of conceptual structures by allowed representation of procedures as recursive dataflow graphs (for a review, see [18]). In these graphs, conditionals are represented by a three-input actor that selects between the two possible concept nodes according to the boolean value of the third input. However, there is no attempt to unify declarative and procedural aspects. Lukose [9] used the idea that every graph that contains an ACT has a procedural counterpart that explains how the ACT is realized. The specification of the procedure used a special scripting language that referenced the concepts and relations in the graph. These procedures, also called actors, followed a message-passing regime in a data-flow model. Again, there was no attempt to unify the representation of declarative and procedural aspects of the visual notation.

In MODEL-ECS, Lukose [7], used declarative CG counterparts to traditional programming language constructs, such as if-then-else, while-do, repeat-until, for-to-do, and case.

Mineau [[10][11]] has embarked on an effort that can be seen as a way of extending the idea of a functional actor into the same realm as Lukose's general-purpose actors. Mineau's processes are like the actors in MODEL-ECS, but are specified as a sequence of transitions between states. Each transition with its pre- and post-conditions is represented as a conceptual graph. This effectively unifies declarative and procedural aspects, but the cost is that every process has to be described in minute detail at every step using the CG counterparts of assignments, conditional testing and operations. In [11] the idea of a constraint is added to the process representation to allow dynamic constraint handling. Thus two processes can communicate their results to each other and alter their behavior accordingly. Again, the new notation is an extension of the existing formalism, preserving declarative and procedural aspects of the knowledge.

5. Visual Programs

Computer programs are representations of a sequence of events (machine code instructions), where each event can be assumed to execute atomically. Clearly, programs have the necessary dynamic characteristics to support procedural knowledge using the foundations of programming languages. Most programming languages, however, are text-based and the understanding of a program relies on language processing. Standard reading of a program says that events happen sequentially unless it is subverted by a special control flow mechanism, which allows a jump from one internal page of memory to another. Text-based languages preclude the advantages offered by visual representation and there is no direct representation of control flow.

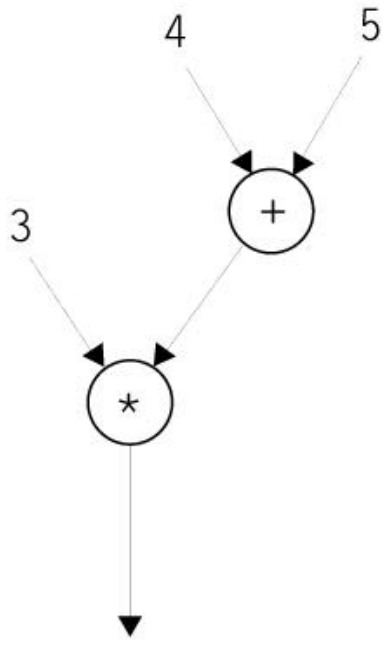

Figure 5: A Simple Data-Flow Expression

In fact, even if the event of jump is defined, it is with a declarative construct, not procedural. General visual languages [5] add a visual construct that can be used as a procedural representation of control flow. This construct come from data-flow languages [1] and uses a functional model plus execution on valid pre-conditions to define the control flow of an expression. Figure 5 shows a simple data-flow diagram of an arithmetic expression 3 * (4 + 5). In this example, each operation is executed only when all its inputs are present. The multiplication operation thus cannot execute until the addition operation is complete. Sequencing of operations is thus ensured because of the dependencies between operations. In most data-flow models, inputs from other operations are consumed by the next operation, which precludes the execution of the operation again until fresh inputs are provided. In this pure data-flow model, there is no program state since there are no variables to retain their values. Text-based functional languages like Lisp or ML have an added feature to give variable-like facilities. However, visual languages make this control flow more attractive because they become a candidate for representing procedural knowledge.

This advantage, however, does not come without drawbacks. Although linear or even parallel sequences of operations are well represented [1], data-flow models have trouble with conditionals and loops. There is, however, an existing visual representation known as *flowcharts*, which use the box and arrow notation to indicate control flow in a computer program. Showing conditionals and loops in this representation is no problem; so notating a jump from one machine command to another can be represented by embedding a test. If we take these two representations and borrow the branching idea from flowcharting and adapt it to the data-flow model, we get a visual representation that can handle control flow including conditionals and loops by flowing data one of two ways, instead of actual control. In Figure 6, we show how this can work for a test involving equality to zero. In this example, Boolean values are passed as inputs to a `special' gate operation that passes its input to one of two outputs depending on the value of the Boolean input. The same idea can be done with loops.

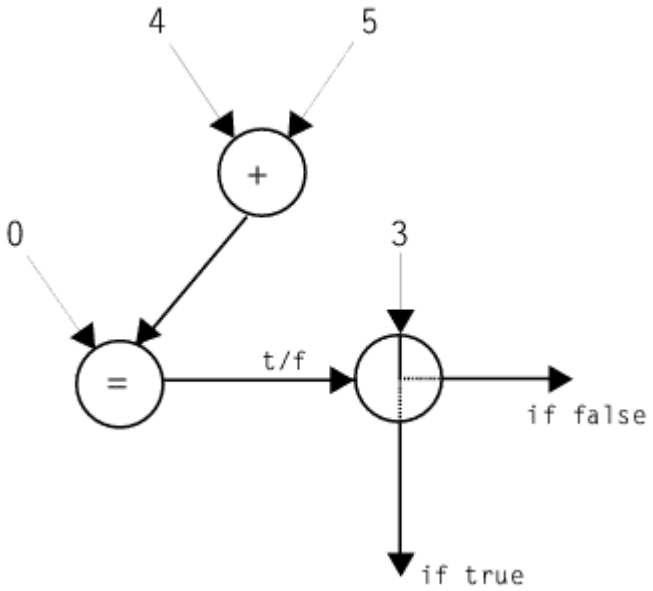

Figure 6: A Data-Flow Conditional

6. New Visual CP

In this new visual implementation, CP uses spatial relationships to support the uniqueness of objects. Two concept nodes in the same graph that contains the same type field, but no referent field, are assumed to represent two different instantiations of the same conceptual idea. To have two different relation nodes, the nodes may be named (given a referent) in order to demonstrate different instances of a relationship. Relation nodes do not relying on relative placement of the object nodes to carry any meaning. The changing of other nodes in the graph implicitly represents temporal actions. Unfortunately, this is rather like writing a program just by specifying the contents of variables only, leaving the assignment of actions to be implied. Perhaps it would be better to represent time with a third dimension, using the visualization of program execution here [14].

Current advancements to CP follow from the preceding discussion. Drawing from the idea purposed by Sowa in 1998 [18], if we take our extended CG representation, the previous CP, and merge it with the data-flow/flowchart representation discussed in section 5, the resulting visual representation language captures both type of knowledge in one notation.

Let us work through an example, looking first at a typical declarative representation of an action in CP, and then examining how it can be modified to the new notation. We chose the easy to understand task of making pancakes. Our aim will be to represent this in such a manner that all the relevant parts of the task are defined declaratively, but the actions use the new procedural representation for execution of the conceptual program. A complete analysis would involve many graphs with agent, patient, source and destination, and perhaps others, for each graph but we will not go into that level of detail for this example.

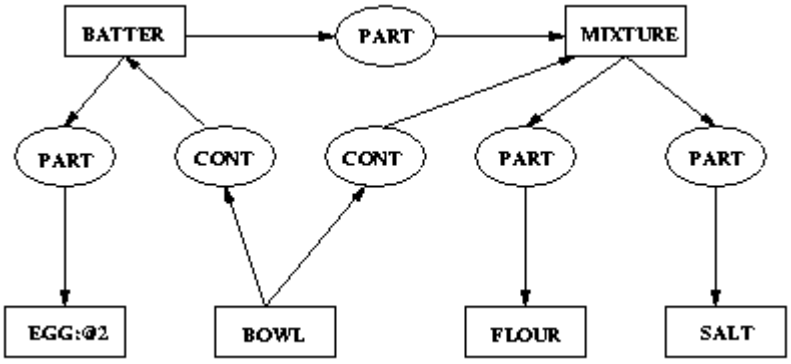

Figure 7: A CG Graph for BATTER

Our first definition looks at the concept of mixing objects into a bowl to make a **BATTER**. In Figure 7, is presented a CG graph for statically declaring this concept. However, there is no representation in this graph to allow the reader to know the correct sequence of the operations, i.e. first mix the flour and salt in the bowl and then add in the eggs. The graph should be broken down into at least two acts; 1) the act of pouring the flour and salt into the bowl and 2) the act of adding in the eggs. Therefore, these actions (seen in Figure 8) will be represented by the actors **POUR** and **ADD** using the extended CG representation discussed in section 4.2.

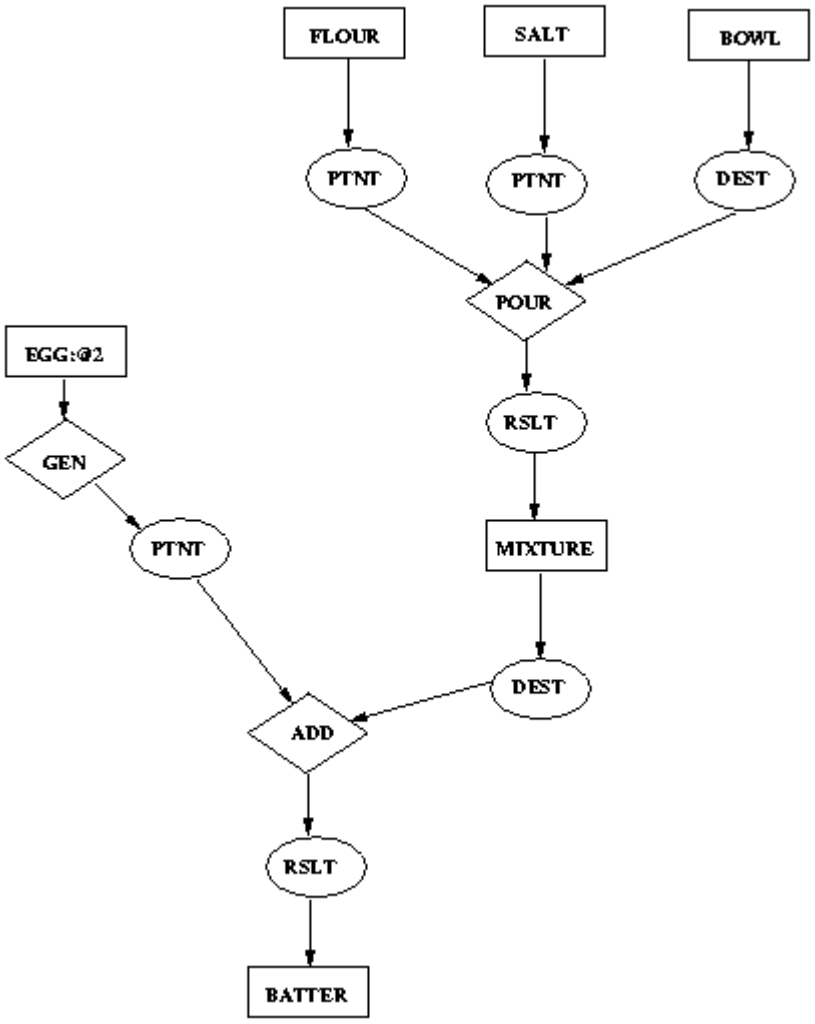

Figure 8: CP Actors Overlays

Next we will need to decide what data is connected in the graph to the given action and how processing flows through the graph. If we employ the RSLT relation, *result*: link an act to an entity that is generated by the act [17], as a generic output case relation, we get the result of putting a **MIXTURE** and eggs into a bowl is to make a **BATTER** (see Figure 8).

Let us now examine the breaking of two eggs into the mixture. This will be done by using a generative construct of putting in one egg, mixing, and putting in other egg. Previously in CP, we would have used a time actor **MIX** [13]. Now we will use a data-flow loop (not shown) that is the CP graph for the actor **GEN**. Next we will have to stir the batter for one minute, and if too thick, add a tablespoon of water. In Figure 9, we show how the visual representation would look if we just merged the declarative representation of procedural knowledge (see in Figure 8) currently in CP with our new visual representation from section 5.

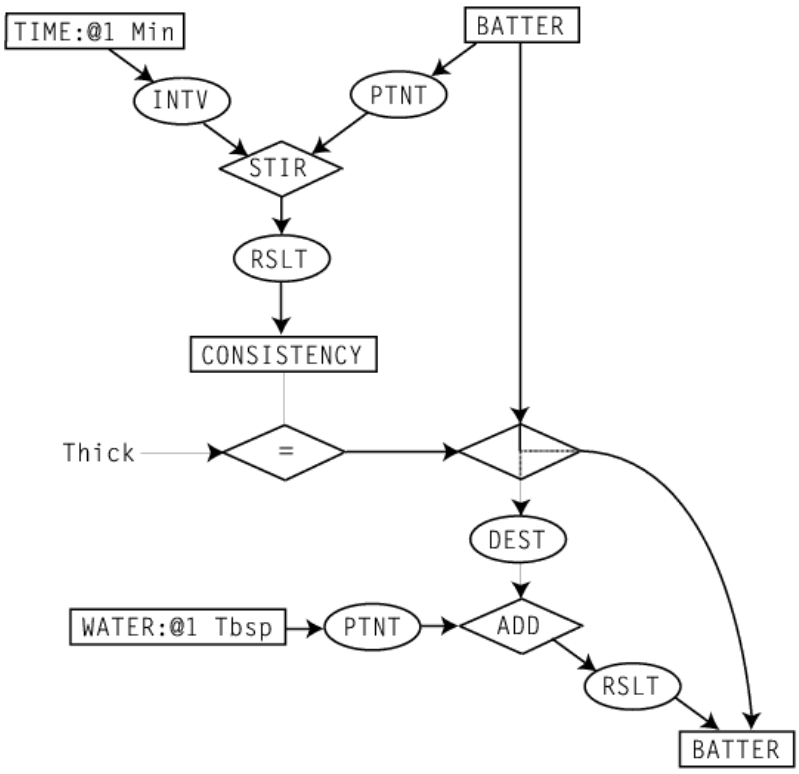

Figure 9: The Gate Action of STIRring

As a point of clarification, the BATTER concept that appears twice in Figure 9 is in fact the same concept. Its referent is however a graph in which the attributes of the graph are the input and output parameters of the action of STIRring and change when the graph is processed. Now the batter is ready for cooking.

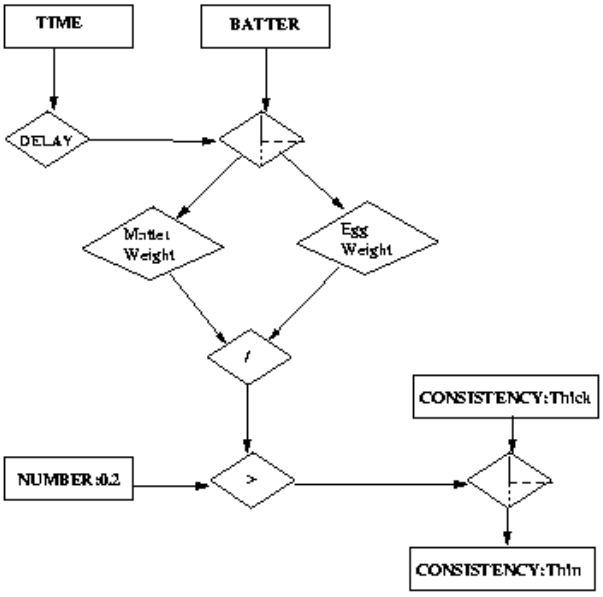

Figure 10: Procedural Action of STIR

If we go back and re-examine the process, we can take it one step further. The graph in Figure 9 describes what is involved with the action of stirring the batter, that is the control flow, but it does not say `how it is to be done'. In order to complete the balancing between the two forms of knowledge, we need a definitional mechanism for actions that are not declarative, such as the action **STIR**. Just as a definition based in spatial snapshots offers a detailed view of a concept in terms of other objects, by using a procedural definition of a temporal snapshot one can see how values can be computed by an action. A procedural definition of **ADD**, **POUR** and **STIR** can be defined in terms of functional actors and data-flow operations, essentially producing a procedure to define not only the control flow, but the how. Figure 10 is an example of the new **STIR** actor with a real procedural definition. However, as the reader will note, there are actor nodes feeding input to other actor nodes. We take the liberty of performing this short hand when the output type from one actor node is the same as the input type to the next actor node. A clear example of this notation can be seen in Figure 11, such that, the output type of the first actor is a number, and the input to the second actor is also the type number.

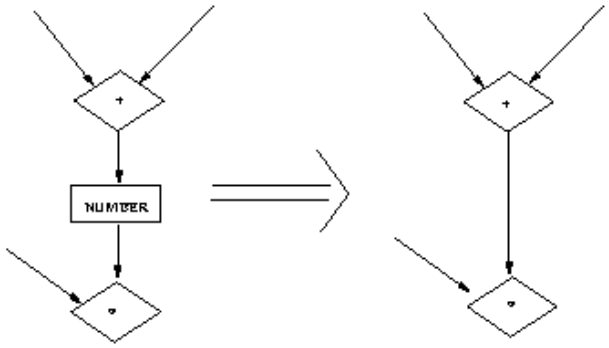

Figure 11: Actor Input and Output Reduction

7. Conclusions and Future Work

We have shown a visual language, CP, that gives a balanced knowledge representation implementation. Within CP, defining object and action constructs equally represents declarative and procedural knowledge. In particular the addition of visual language constructs (data-flow/flowchart) to CP actor overlays allow it to process actions as data-flow diagrams conveying the procedural nature of the knowledge within the representation.

Details not addressed are actually handling the nature of time (moments vs intervals, events vs processes, etc.) and the procedural actions of all the actors in the CP graphs. We also have not presented the details on presenting and then converting to the visual knowledge representation for things such as time charts to the new CP actor overlays. Automatic display generation of conceptual graphs is important for validation of the graphs and communication of the knowledge within these graphs [2]. These will be examined in future works.

References

[1] Auguston, M. and A. Delgado (1997) Iterative Constructs in the Visual Data Flow Language, in Proceedings of IEEE Symposium on Visual Languages, Isle of Capri, Italy.

[2] Cyre, W.R., S. Balachandar and A. Thakar (1994) Knowledge Visualization from Conceptual Structures, in Lecture Notes in AI Vol 835 Conceptual Structures: Current Practices, eds. W.M. Tepfenhart, J.P. Dick and J.F. Sowa. New York:Springer-Verlag.

[3] Davis, R. (1980) Meta-rules: Reasoning about Control, in Artificial Intelligence Vol. 15 No. 3.

[4] Dean, T. (1983) Time Map Maintenance. Tech. Rep. 289. Yale University Computer Science Department.

[5] Glinert, E. (1990) Visual Programming Environments, (2 Vols). Los Alamintos, CA: IEEE Computer Science Press.

[6] Hartley, R.T. (1992) A Uniform Representation for Time and Space and Their Mutual Constraints, in Computers Matt. Applic. Vol. 23 No. 6-9, pp.441-457.

[7] Kremer, R., D. Lukose, and B.Gaines (1997) Knowledge Modeling Using Annotated Flow Chart, in Lecture Notes in AI Vol. 1257, New York:Springer-Verlag.

[8] Lin, S. H. and T. Dean (1994) Exploiting Locality in Temporal Reasoning, in Current Trends in AI Planning, eds. E. Sandewall and C Backstom. Amsterdam:IOS Press.

[9] Lukose, D. (1993) Executable conceptual structures, in Lecture Notes in AI Vol. 699, New York:Springer-Verlag.

[10] Mineau, G. W. (1998) From Actors to Processes: The Representation of Dynamic Knowledge Using Conceptual Graphs, in Lecture Notes in AI Vol. 1453, New York:Springer-Verlag.

[11] Mineau, G. W. (1999) Constraints on Processes: Essential Elements for the Validation and Execution of Processes, in Lecture Notes in AI Vol. 1640, New York:Springer-Verlag.

[12] Pfeiffer, H.D. and R.T. Hartley (1991) The Conceptual Programming Environment, CP: Reasoning Representation using Graph Structures and Operations, in Proceedings of IEEE Workshop on Visual Languages, Kobe, Japan.

[13] Pfeiffer, H.D. and R.T. Hartley (1992) Temporal, Spatial, and Constraint Handling in the Conceptual Programming Environment, CP, in Journal for Experimental and Theoretical AI Vol. 4 No. 2, pp.167-182.

[14] Price, B.A., R.M. Baecker and I.S. Small (1993) A Principled Taxonomy of Software Visualization, in Journal of Visual Languages and Computing Vol. 4 No. 3, pp. 211-266.

[15] Rochowiak, D. M. (1997) A Pragmatic Understanding of "Knowing That" and "Knowing How": the Pivotal Role of Conceptual Structures, in Lecture Notes in AI Vol. 1257, New York:Springer-Verlag.

[16] Ryle, G. (1949) The Concept of Mind. Harmondsworth, UK: Penguin Books.

[17] Sowa, J.F. (1984) Conceptual Structures. Reading, MA: Addison-Wesley.

[18] Sowa, J.F. (1998) Conceptual Graph Standard and Extensions, in Lecture Notes in AI Vol. 1453, New York:Springer-Verlag.