

Conceptual Integration In Multiple Viewed Requirements Development

Harry S. Delugach

Computer Science Department

Univ. of Alabama in Huntsville

Huntsville, AL 35899 U.S.A.

phone: (205) 895-6614

fax: (205) 895-6239

Email: delugach@cs.uah.edu

WWW: <http://www.cs.uah.edu/~delugach>

Abstract

This paper addresses software requirements development, and how it can be supported by combining multiple views of participants with the ability for the participants to gain feedback from other views. We include a brief justification for the inclusion of multiple views, a brief summary of multiple-viewed approaches, and introduce conceptual graphs as a representation method for requirements. We use some simple techniques, using a brief example, along with the Wordnet database to show how conceptual feedback supports elicitation and acquisition. We finally outline some future work we will undertake to further explore the techniques.

Keywords: software requirements engineering, requirements acquisition, requirements elicitation, conceptual graphs, conceptual feedback.

Conceptual Integration In Multiple Viewed Requirements Development

1. Introduction

This paper addresses software requirements development, and how it can be supported by combining multiple views of participants with the ability for the participants to gain feedback from other views. There is now widespread agreement that software requirements play a crucial role in developing effective and reliable software applications [5]. A variety of techniques to enhance requirements have been developed over the years: object-oriented analysis [3] [32], rapid prototyping [2] [33], etc.

Another current notion in computer system development is the acknowledgment that multiple participants (“stakeholders”) are necessarily involved in systems development, and that multiple views of the stakeholders must be accommodated in some fashion into the requirements [14] [22] [23] [27] [28]. Communication between the stakeholders (particularly including customers) also leads to more successful software being delivered [20].

A third current trend is to focus on the *process* of software development, as well as its products. A prime example is the CMM (Capability Maturity Model) [17] [18]. Underlying these efforts is the assumption that an effective development process will produce a quality product. Considering a requirements specification as a product in that sense, researchers and practitioners alike have focused on understanding and improving the software requirements development process [5] [15].

This work follows in the spirit of others in pursuing a synthesis of these three notions — leveraging requirements, incorporating multiple views, and focusing on the process — in order to gain a better understanding of system’s requirements. We first justify the inclusion of multiple views in the *process* of software requirements development, then explain conceptual feedback as a means of reaching early agreement on some concepts. A brief example illustrates the flavor of the technique. We then summarize our results and point to future work.

2. Why Multiple Views?

In recent years, more and more attention has been given to problems in requirements development that can be effectively addressed by acknowledging the importance of multiple views. Researchers no longer have to argue the efficacy of explicitly including and defining multiple views. There is a body of work, as suggested by a recent ACM workshop [1] to support further pursuit and development of multiple viewed approaches. This section summarizes the current state of multiple-viewed research and suggests enhancements to the current approaches.

In this work, we assume that a view originates with an individual person, typically as a result of their job responsibility in the context of a project. Our terminology follows from Leite’s [23]. We call a person involved in requirements development a *participant* in the development. Their particular job responsibility constitutes a *viewpoint*, e.g., the implementer’s viewpoint or the user manual author’s viewpoint. Within a viewpoint, there may be multiple *perspectives*; e.g., the debugging perspective of an implementer, or the portability perspective of the user manual author. Given these three things: a participant, a viewpoint and a perspective, a *view* can be created. In general, a view can be any representation or description of any part(s) of a system from a given participant’s viewpoint and perspective.

Many recent researchers have acknowledged the presence of multiple views, and a few have even explicitly modeled them as distinct views. Some representative work is as follows. The work of Nissen, et al [27] is an example of a practical technique that is used in commercial settings to form a framework for discussion and negotiation among participants. Its biggest drawbacks are (a) it depends upon having a skilled (human) facilitator, thus allowing for possible biases to appear and (b) there is no formal way of modeling negotiation or overlap. Viewpoints [14] [28] describes a partitioning of each viewpoint to provide a small number of categories for viewpoint content. These categories have been determined manually to be useful in examining multiple viewpoints for consistency and completeness via a rule-based approach. The work of Leite [23] addresses the question of eliciting different opinions about what a system’s requirements are, and supports the negotiation process whereby the differing opinions may be reconciled. The terminology and framework of Leite’s work are related to the current proposal. Leite’s main focus is in providing an example of how viewpoints may be reconciled, again by using rule-based models. The main drawback of rule-based models is their lack of flexibility.

Another related approach is that of van Lamsweerde, who uses semantic networks as one of three formalisms for capturing multiple views [21] [22]. This paper proposes an approach that can be supported by conceptual graphs, based on semantic networks and predicate logic, and does not require any other representations, thereby offering the opportunity for a single unified internal representation. Some extensions have already been made to the original conceptual graph formulation in [34] that address issues of non-monotonic logic and second-order logic [6] [7].

3. Are Multiple Views Really Separate Views?

A major problem with previous approaches to multiple viewed software requirements (including my own [7] [8]) is that they often assume that separate multiple views have somehow been acquired and are then to be analyzed for their consistency, conflicts, etc. (e.g., [16]) To be sure, there are probably a few environments in which such a situation holds; e.g., a company that has several divisions involved in development, or more likely, a project being supported by several different development organizations, each with their own methodology, corporate culture, personnel experience, and so on. The time constraints that exist in most modern system development often dictate that the separate stakeholders develop their system and software requirements in parallel, and that their overlap, inconsistency, conflicts, etc. be handled at the end.

Producing multiple views separately and then integrating them is an attractive approach from an organizational or managerial standpoint. Each view can be produced more-or-less independently, with each organization's management responsible for producing its own version of its portion of the system's requirements. Managers can therefore perceive that they retain substantial control over their view of the requirements.

Such an imposed compartmentalization is clearly an illusion, however. The requirements of a large system are dependent (and inter-dependent) upon the needs and wants of a diverse group of individuals, and an arbitrary partitioning of the requirements at the outset (even if performed carefully) does not change those needs and wants. A more effective approach to multiple viewed requirements is, therefore, to acknowledge at the outset that each organization's efforts and products will be affected by the other organizations' efforts and products. If the effect of that interaction is accommodated early and often in the development process, then each stakeholder's efforts can be kept "on course" and stray as little as possible from the shared expectations of the eventual system.

This paper describes a concept I believe to be central to obtaining this early accommodation of multiple views, a concept I have dubbed "conceptual feedback" whereby stakeholders learn about each other's views and develop a shared understanding (one might want to say a "shared view") while they are developing their respective views. Some earlier ideas along these lines are found in [6] and [10]. An important aim of this research in dealing with shared views is that the collective requirements will be representable as a whole. That is, we intend to capture all requirements in a repository such that they are all represented in a common notation. We have chosen the representation of conceptual graphs [34].

Not all researchers agree that there should be a single repository of requirements knowledge. For example, Nuseibeh and Finkelstein [28] have developed an approach whereby features of each relevant viewpoint are mapped directly to features in other viewpoints. This has some advantages, e.g., not all features in every viewpoint are relevant to features in every single other viewpoint, and it also allows tailoring of specific constraints within each viewpoint. Two disadvantages, however, are clear:

- (1) For N viewpoints, such an approach requires $(N^2 - N) / 2$ separate mapping schemes, each requiring its own analysis, development and validation. Assuming that separate mapping schemes are necessary in both directions, the number is doubled to $N^2 - N$. Given that the number of viewpoints is at least four or five people in a typical software system's development, the analysis of viewpoint interactions become unwieldy and not cost-effective.
- (2) Pre-defining features of the overlap may neglect additional interesting features of the overlap that may be useful in reaching agreement among the participants, and restrict the flexibility of the approach.

We have therefore proposed a common repository such that mappings are necessary only between each viewpoint's features and the repository's language (and in reverse). This means that the number of mapping schemes for N viewpoints is only $2N$, even when both directions (to and from the repository) are considered. The N^2 scheme could still be used to alleviate shortcomings in the $2N$ scheme if necessary, but the cost-effectiveness of a multiple-viewed approach is enhanced by first applying a $2N$ approach.

We also adopt a knowledge-based approach whereby diverse kinds of information can be represented and manipulated, and therefore having the potential of reduced bias and the ability to uncover new and/or unexpected interactions between views. This knowledge-based approach is centered around the principles of logic and semantic networks. For our common representation, we have chosen conceptual graphs, a semantic network representation based on first-order logic as originally proposed by the philosopher C. S. Peirce. Conceptual graphs have a rich base of previous study [11], [12], [13], [25], [29], [34], [36] and have already been used in representing information systems concepts [4] [9]. Indeed there is some previous work in applying conceptual graphs to computer system development [6] [7] [19] [31]. We will not go into details of conceptual graphs here, for a concise introduction, see [30] or [35].

4. Supporting A Multiple-Viewed Analysis Process

van Lamsweerde has identified three important problems in resolving differences between views [21]. There are, however, several other issues that can arise. Consider that in developing specifications, stakeholders use names for things. There are a number of difficulties encountered just in handling names in a multiple-viewed context. These problems include:

- (a) A “synonym problem”: where two persons each use a different term, but each means the same thing.
- (b) A “homonym problem”: where two persons each use the same term, but each has a different meaning.
- (c) A “unknown word problem”: where one person uses a term or concept that the other does not know.
- (d) A “subclass problem”: where one person’s term or concept is a subclass (subtype) of the other person’s.
- (e) A “granularity problem” : where one person’s term or concept is an aggregate of several terms or concepts of the other person.
- (f) A “generality problem” : where one person’s term is a generalization of another’s (e.g., one says *operator*, the other says *person*)
- (g) An “instance” problem : where the two persons appear to agree on a term, but still intend for it to mean different instances of the same thing.

The approach we describe clearly addresses problems (a), (b) and (d), and also problem (c) to a limited extent. Although there are many features desired of a requirements representation for knowledge based support, I will use two simple ones here in order to illustrate the process of conceptual feedback with human users. The two features are concept definitions and an inheritance hierarchy of concept types.

Rather than construct a “toy” conceptual type hierarchy for the purposes of this paper, I have used Wordnet, a database of word meanings and their position in a type hierarchy [24]. In Wordnet, a word is classed as one or more parts of speech (i.e., noun, verb, or adjective) and within each part of speech, a word’s meaning can be further divided as to its “sense,” thus effectively providing homonyms. The Wordnet database is an excellent illustration of how a simple structure, when populated with a substantial amount of information, can serve as a foundation for knowledge-based requirements development. This paper uses Wordnet to simulate an actual conceptual catalog with definitions and a hierarchy.

4.1 Elicitation and Acquisition

We present two brief examples of how simple techniques with a rudimentary underlying knowledge base can still support an effective means of eliciting and acquiring requirements using conceptual feedback. In these examples, we do not identify an explicit participant; we need only assume there is a real person with whom an automated system can interact.

The two examples are centered around developing requirements for a patient monitoring system, an example often used to illustrate development techniques. The first example is a UML object diagram shown in Figure 1. Once the diagram is created, we assume an automated system that extracts its main features, queries a Wordnet database for additional knowledge to incorporate, and from there translates the diagram into conceptual graphs. The results of the query are shown in Figure 2, and a conceptual graph (with a hierarchy) is shown in Figure 3.

Since this is an object-oriented diagram, the names of each object will be assumed to be nouns. A noun search of each name using Wordnet produces the information in Figure 2 (shown in Wordnet’s default output form). The shaded portions will be rejected by the participant in a brief interactive session; the portions in white are to be accepted.

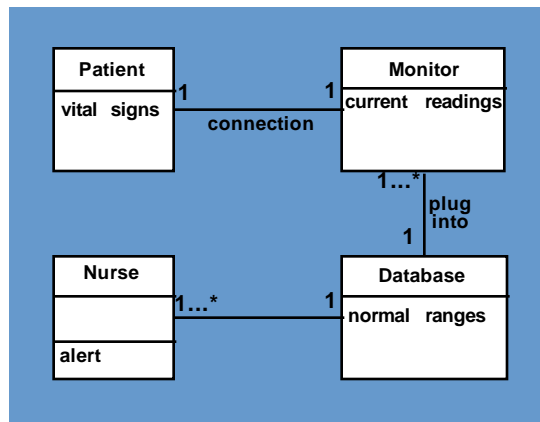


Figure 1. Originating Object Diagram in UML.

Assuming that this information was presented to the participant who wrote the UML diagram in Figure 1, we show that the person validates the incorporated knowledge by putting a check (✓) next to the sense they have chosen. Since “patient” has only one sense (as a noun) and “database” has only one sense, the participant has the choice to either accept or reject the incorporated refinements.

Since the word “nurse” has two senses, the participant must choose between the two (or reject both). In this case, since the patient monitoring system is to be developed for a hospital setting, the participant chooses the health care professional sense, rather than the “nanny” sense¹.

The word “monitor” has several senses (meanings) from which the participant can choose (or reject). There are actually three senses that might be relevant: sense 3 (e.g., a computer display), sense 4 (e.g., a sensing device) and sense 5 (a regulator). A person may easily conclude that sense 3 is probably not what he/she intended; however, sense 4 and sense 5 both represent devices that could be hooked up to a patient in a hospital. Sense 4 represents a passive information-gathering device, whereas sense 5 represents a controller of a patient. This choice is not just a matter of semantics! It is instead an opportunity for the person to decide and then declare whether the monitor is to be passive or not. In this example, the person chose sense 4, the monitor as an information collector.

We point out that the choice for the word “monitor” actually involved two distinct processes — namely, *elicitation* and *acquisition* — although the boundary between them may not always be clear. Presenting alternatives from which the person chooses (e.g., sense 4 and sense 5 of monitor) constitutes a process of *elicitation*, i.e., assisting a person in deciding what they really want. Recording the person’s choice (e.g., they choose sense 4) is a process of *acquisition*, namely capturing the information needed to express requirements. This simple example illustrates the difference between these two processes.

Diagrams from different paradigms can be represented under the single formalism of conceptual graphs. For example, Figure 1 is expressed as the conceptual graph in Figure 3 (see [7] for details).

¹Note that Wordnet’s database is a descriptive one, not a proscriptive one, so that included words and meanings are obtained from large samples of writing in the English language. It therefore does not deal with gender bias in language, nor does it distinguish between casual usage and formal usage, etc. except that it presents word senses more-or-less in order of frequency.

patient (noun)	monitor (noun)
<p>patient, case ✓ => sick person, sufferer => unfortunate, unfortunate person => person, individual, someone, mortal, human => life form, organism, being, living thing => entity => causal agent, cause, causal agency => entity</p>	<p>Sense 1 proctor, monitor => supervisor => superior, higher-up, superordinate => leader => person, individual, someone, mortal, human => life form, organism, being, living thing => entity => causal agent, cause, causal agency => entity</p>
nurse (noun)	<p>Sense 2 admonisher, monitor, reminder => defender, guardian, protector => person, individual, someone, mortal, human => life form, organism, being, living thing => entity => causal agent, cause, causal agency => entity</p>
Sense 1	<p>Sense 3 monitor => display => electronic device => device => instrumentality, instrumentation => artifact, artefact => object, inanimate object, physical object => entity</p>
<p>Sense 2 nanny, nursemaid, nurse => woman, adult female => female, female person => person, individual, someone, mortal, human => life form, organism, being, living thing => entity => causal agent, cause, causal agency => entity => custodian, keeper, steward => defender, guardian, protector => person, individual, someone, mortal, human => life form, organism, being, living thing => entity => causal agent, cause, causal agency => entity</p>	<p>Sense 4 monitor ✓ => electronic equipment => equipment => instrumentality, instrumentation => artifact, artefact => object, inanimate object, physical object => entity</p>
database (noun)	<p>Sense 5 monitor => regulator => control, controller => mechanism => device => instrumentality, instrumentation => artifact, artefact => object, inanimate object, physical object => entity</p>
<p>database ✓ => information, info => message, content, subject matter, substance => communication => social relation => relation => abstraction</p>	<p>Sense 6 monitor, monitor lizard, varan => lizard => saurian => diapsid, diapsid reptile => reptile, reptilian => vertebrate, craniate => chordate => animal, animate being, creature, fauna => life form, organism, being, living thing => entity</p>

Figure 2. Type hierarchy chosen from object diagram.

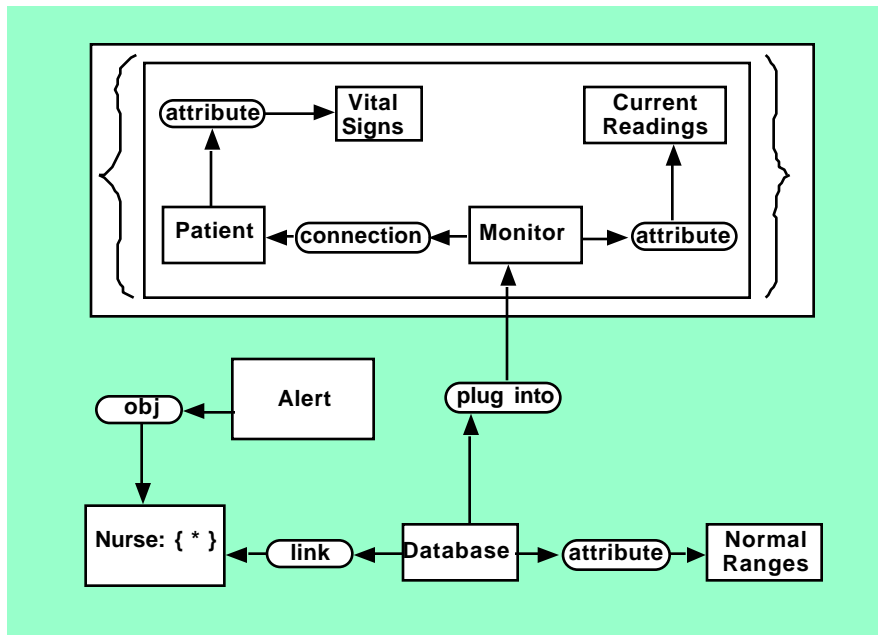


Figure 3. Conceptual graph representation of UML diagram.

Although the previous example is simple, it illustrates two important features of a true knowledge-based multiple-viewed approach:

- Decisions are made based on choices presented from a pre-existing knowledge base (i.e., Wordnet), and
- The decisions are captured in a view-independent form with which other views can share. That is, other views will operate on the same underlying knowledge base, with the same numbered senses of the same words.

As a second example, consider the data flow diagram in Figure 4.

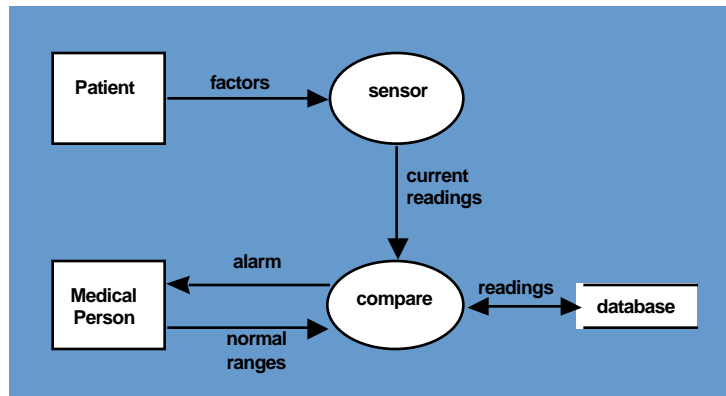


Figure 4. Originating data flow diagram.

patient (noun)	compare (verb)
patient, case ✓ => sick person, sufferer => unfortunate, unfortunate person => person, individual, someone, mortal, human => life form, organism, being, living thing => entity => causal agent, cause, causal agency => entity	Sense 1 compare ✓ compare, draw a comparison between => analyze, analyse, study, examine
sensor (verb)	Sense 2 compare => be, have the quality of being
<no verb sense found> <noun sense found>	Sense 3 compare, liken, equate, consider equal => see, consider, take to be, reckon, view, regard => think, believe => judge, form an opinion of, pass judgment on
database (noun)	medical person (noun)
database ✓ => information, info => message, content, subject matter, substance => communication => social relation => relation => abstraction	<no noun sense found>

Figure 5. Type hierarchy chosen from data flow diagram.

This participant was presented with the same type hierarchies for “patient” and “database” as the previous one, and makes the same choices. Although this choice may not seem to provide new information, in fact it does: namely, that the two participants agree on those two senses of the words in their respective descriptions of the system. Areas of agreement are important, both for producing a final set of requirements, and in serving as a basis for further discussion about concepts on which there may exist disagreement.

Some parts of the second example are more interesting than the first. Note that in two cases, the desired term is not found. Intuitively, we may suspect that the term “sensor” is not a verb and is therefore perhaps an inappropriate name for a process. The term “medical person,” as a compound term, would only appear in the database if it were in common usage. We can adopt strategies to deal with cases such as these, such as:

- Find the word as a different part of speech.
- Find synonyms of the word as a different part of speech and identify those which have senses in the original part of speech (e.g., the verb “sensor” is looked up as a noun and its synonyms checked as verbs).
- Look up component terms of a compound (i.e., multi-word) term.
- Get a new word from the participant.

We can adopt the following algorithms as a procedure for accommodating these strategies.

```

begin ConfirmWords
  foreach word W unmarked in diagram D
    get W's part of speech POS from a semantic description of D
    confirmedword = ConfirmOne ( W, POS )
  end foreach
end ConfirmWords

```

```

begin ConfirmOne ( word W, partofspeech POS ) returns word
  boolean marked = false;
  repeat
    look up POS( W )
    if pos( W ) found then
      display senses 1 through N
      get choice from user
      case choice of
        n :      mark W with sense 1
              return W;
        none :   call FindAnother ( W )
        alternative : ConfirmOne (alternative)
      end case
    else
      if compound term w = ( First, Second ) then
        if POS ( Second ) found then ConfirmOne ( Second, POS )
        if adjective( First ) found then ConfirmOne ( First, Adjective )
      else
        word New = FindAnother( W )
        ConfirmOne ( New )
      end if
    end if
  until marked or user_skips;
end ConfirmOne

```

```

FindAnother (word w) returns
  find w as some other part of speech
  display synonyms of w that match pos(w)

  display coordinate terms of w that match pos(w)

  display definition; search for words that match pos(w)
  query user for another name
  suggest a name from the rest of the specification
end FindAnother

```

Figure 6. Algorithms For Eliciting Terms From A Participant.

In the example of Figure 5, the term *sensor* is recognized as a noun, but not as a verb, and *medical person* was not found. We can thus illustrate two of the strategies included in the algorithm.

If a term is not recognized as the part of speech desired, then assume it is a synonym for another term that is known as that part of speech. In this case, the strategy calls for seeking the term *sensor* as a different part of speech, identifying synonyms of those terms, and then seeing if any of the synonyms are known as nouns.

Synonyms of *sensor* as a noun are “detector” and “device.” Since neither of these is suitable as a verb, we must stop our search here. Though our intuition might tell us to use the verb “detect” as a synonym for the verb “sensor,” we have not yet included such avenues in our approach.

The term *medical person* was not found as any sense, although Wordnet does include a number of compound terms. As a compound term, however, we can make an initial assumption (in English) that its last term *person* is a noun, and its preceding term *medical* is either an adjective or adverb. Figure 7 shows the results from a Wordnet search:

person (noun)	medical (adjective)
Sense 1 ✓ person, individual, someone, mortal, human, soul => life form, organism, being, living => causal agent, cause, causal agency	Sense 1 ✓ medical Pertains to noun -> medicine (Sense 3)
Sense 2 person => grammatical category, syntactic category	Sense 2 medical (vs. surgical) -- (requiring or amenable to treatment by medicine esp as opposed to surgery: "medical treatment"; "pneumonia is a medical disease")
Sense 3 person => human body, physical body, material body, soma, build, figure, physique, anatomy, shape, chassis, frame, form	Sense 3 aesculapian, medical -- (of or belonging to Aesculapius or the healing art)
	<hr/> medicine (noun - Sense 3) <hr/>
	Sense 3 medicine, practice of medicine => learned profession

Figure 7. Possible type hierarchy.

The result of all these queries is a set of common terms that the two participants have agreed upon. In conceptual graph terms, we say that we can *join* the two participants' graphs around the common terms. Such a join produces the graph in Figure 8. The upper graph is obtained from the data flow diagram in Figure 4. The dashed lines represent "lines of identity" between co-referent concepts; i.e., concepts that represent the same individual. The two graphs therefore become one, as each pair of co-referent concepts collapses into a single concept. For simplicity and brevity, we did not undergo the elicitation/acquisition process for any attributes or data flows; only those concepts agreed-upon so far are joined.

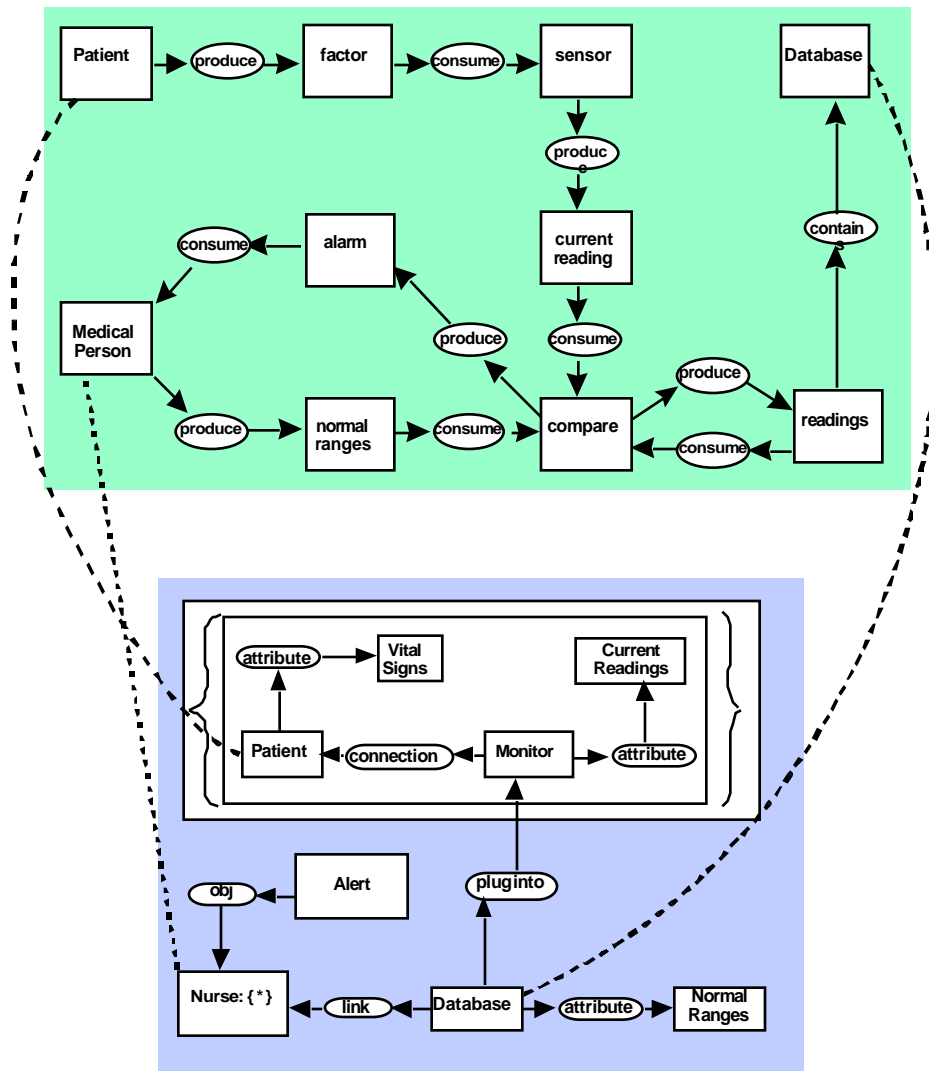


Figure 8. Two Joined Views.

4.2 Producing a Specification

While we believe that understanding of requirements is the key to building quality software systems, we do not wish to imply that a specification document is no longer needed. Like all written expressions, a specification is easiest to create when one has a clear notion of what one wants to say. A good way to determine how well the requirements are understood is to create a requirements specification that captures one's current knowledge about the requirements. If such a document cannot be produced, then clearly some fundamental knowledge is lacking.

In the limited scope of this work, only a small part of a complete requirements specification can be documented, namely a glossary of terms and their definitions. We consider these to be important first steps in building a complete specification; however, as a product in themselves, they would constitute only a section of the complete document. Paraphrasing a conceptual graph is generally straightforward; see [10] for examples of how such graphs would be portrayed in English.

Given that the terms are obtained from an inheritance hierarchy in the first place, the specification process can include the presentation of a derived ontology, using only the classes of the terms that were used. For this example, the specification's hierarchy would be as shown in Figure 9.

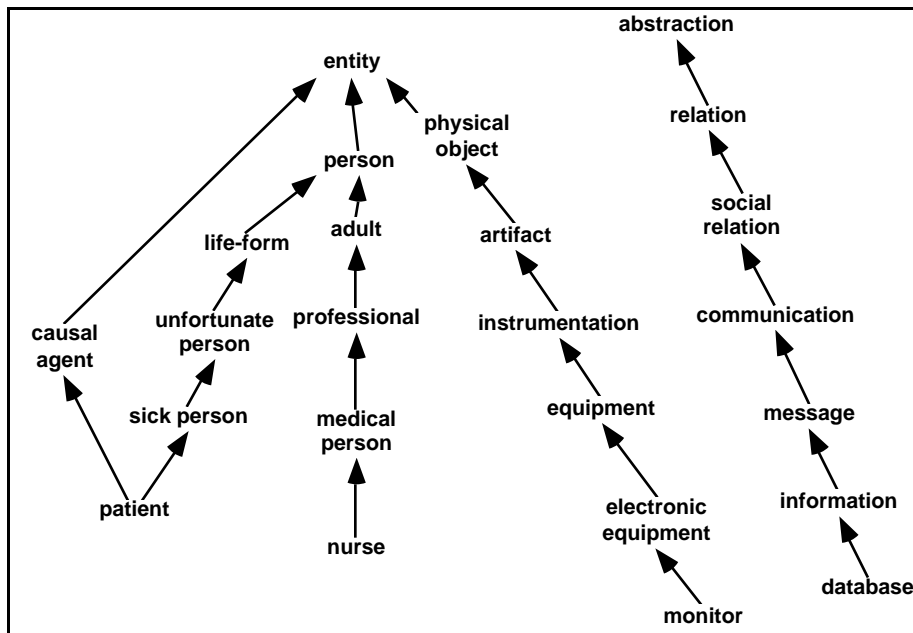


Figure 9. Elicited Type Hierarchy.

In a general sense, producing a requirements specification possesses many aspects of natural language generation problems, and such problems are known to have many difficult and subtle issues. We would be satisfied for the moment to produce a specification that is understandable, without worrying (yet) about the style or flow of that specification.

5. Conclusion

The approach outlined here provides a framework for eliciting and acquiring multiple-viewed requirements. It not only supports multiple representations, but multiple users as well. Obtaining a hierarchy of terms (on which the participants can agree) may be able to support an object-oriented design process; more work is needed in how to elicit more inheritance in the requirements development process.

The limitations of the work are clear: we have dealt only with minimal type definitions and a class hierarchy, probably the easiest and most straightforward knowledge structures. One purpose of this paper is to show that even a fairly limited set of knowledge concepts can aid in feedback during multiple viewed requirements construction. Our future work is to refine the techniques outlined here, and then expand the work into additional representations of requirements that will capture concepts vital to system development, such as the dynamic behavior of processes, algorithms, trade-offs, etc. We have undertaken a study of several widely used requirements specification standards, and we intend to develop automated techniques for producing additional parts of a requirements specification from these additional representations.

Another limitation goes along with using any pre-existing knowledge base in support of requirements elicitation: what if the participant's concepts are new and different from existing ones? What is needed are elicitation and acquisition techniques that will support heretofore unknown ideas. One promising technique is that of tracked repertory grids [37] that is a self-organizing, relatively un-biased, free-form acquisition technique.

One final note: it may be argued that if conceptual graphs can indeed capture all knowledge necessary for supporting multiple viewed requirements engineering, then why do we not simply have all stakeholders use conceptual graphs as their language for requirements? The answer is that stakeholders often have their own prior notations and representations that have served them well in their own views, and conceptual graphs are apt to prove too general and perhaps too low-level for user-oriented expressions. Our purpose is to use them internally as a knowledge representation, not as an interface for user interaction.

6. References

- [1] ACM SIGSOFT, Viewpoints 96: International Workshop on Multiple Perspectives in Software Development, published in *Joint Proc. of the SIGSOFT'96 Workshops*, ISBN 0-89791-867-3, Oct. 14-15, 1996, San Francisco
- [2] Bischofberger, W., and G. Pomberger, eds., *Prototype-Oriented Software Development*, Berlin, Germany, Springer Verlag, 1992.
- [3] Booch, Grady, *Object-Oriented Analysis and Design*, Redwood City, California: Benjamin/Cummings, 1994.
- [4] Carbonneill, Boris and Haemmerle, Ollivier, "Standardizing and Interfacing Relational Databases Using Conceptual Graphs," in Tepfenhart, William M., Dick, Judith P., and Sowa, John F., eds., *Conceptual Structures: Current Practices*, Lecture Notes on Artificial Intelligence LNAI 835, Springer Verlag, 1994, pp. 311-330.
- [5] Davis, Alan M., *Software Requirements: Object, Functions and States*, Prentice-Hall, 1993.
- [6] Delugach, Harry S., "A Multiple Viewed Approach to Software Requirements," Ph.D. dissertation, Dept. of Computer Science, University of Virginia, May, 1991.
- [7] Delugach, Harry S., "Specifying Multiple-Viewed Software Requirements With Conceptual Graphs," *Jour. Systems and Software*, vol. 19, pp. 207-224, 1992.
- [8] Delugach, H.S. "Analyzing Multiple Views Of Software Requirements," in *Conceptual Structures: Current Research and Practice*, Eklund, P., Nagle, T., Nagle, J. & Gerholz, L. eds., 1992, 391-410, Ellis Horwood.
- [9] Delugach, Harry S. and Hinke, Thomas H., "Microanalysis: Acquiring Database Semantics In Conceptual Graphs," in *Conceptual Structures: Knowledge Representation as Interlingua*, P.W. Eklund, G. Ellis and G. Mann, eds., Lecture Notes on Artificial Intelligence LNAI 1115, Springer Verlag, 1996, presented at ICCS '96, 4th Intl. Conf. on Conceptual Structures, University of New South Wales, Sydney, Australia, Aug. 19-23, 1996.
- [10] Delugach, Harry S., "An Approach To Conceptual Feedback In Multiple Viewed Software Requirements Modeling," presented at Viewpoints 96: International Workshop on Multiple Perspectives in Software Development, *Joint Proc. of the SIGSOFT'96 Workshops*, ISBN 0-89791-867-3, pp. 242-246, Oct. 14-15, 1996, San Francisco.
- [11] P.W. Eklund, G. Ellis and G. Mann, eds., *Conceptual Structures: Knowledge Representation as Interlingua*, Lecture Notes on Artificial Intelligence LNAI 1115, Springer Verlag, 1996.
- [12] Eklund, P., Nagle, T., Nagle, J. & Gerholz, L. [Eds.], *Conceptual Structures: Current Research and Practice*, Ellis Horwood, 1992.
- [13] Ellis, Gerard, Levinson, Robert, Rich, William, and Sowa, John F., eds., *Conceptual Structures: Applications, Implementations and Theory*, Lecture Notes on Artificial Intelligence LNAI 954, Springer Verlag, 1995.
- [14] Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L. & Goedicke, M. "Viewpoints: a framework for integrating multiple perspectives in system development," *Intl. Jour of Software Engineering and Knowledge Engineering*, 2 (1), 1992, 31-57.
- [15] Gause, Donald C. and Weinberg, Gerald M., *Exploring Requirements: Quality Before Design*, 1989.
- [16] Heitmeyer, Constance L., Jeffords, Ralph D., and Labaw, Bruce G., "Automated Consistency Checking of Requirements Specifications," *ACM Trans. Softw. Engr. Methodology*, 5(3), July, 1996, pp. 231-261.
- [17] Humphrey, Watts S., "Characterizing the Software Process," *IEEE Software*, Vol. 5, No. 2, March, 1988, pp. 73-79.
- [18] Humphrey, Watts S., *Managing the Software Process*, Addison-Wesley, Reading, MA, 1989.
- [19] Kamath, R.Y., and Cyre, W. R., "Automatic Integration of Digital System Requirements using Schemata," in Ellis, Gerard, Levinson, Robert, Rich, William, and Sowa, John F., eds., *Conceptual Structures: Applications, Implementations and Theory*, Lecture Notes on Artificial Intelligence LNAI 954, Springer Verlag, 1995, pp. 44-58.
- [20] Keil, Mark and Carmel, Erran, "Customer-Developer Links in Software Development," *Comm. ACM*, 38(5), May, 1995, pp. 33-44.
- [21] van Lamsweerde, Axel, "Divergent Views in Goal-Driven Requirements Engineering," presented at Viewpoints 96: International Workshop on Multiple Perspectives in Software Development, *Joint Proc. of the SIGSOFT'96 Workshops*, ISBN 0-89791-867-3, pp. 252-256, Oct. 14-15, 1996, San Francisco.
- [22] van Lamsweerde, Axel, Darimont, R., and Massonet, P., "Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learned," *Proc. RE'95, 2nd Intl. Symposium on Requirements Engineering*, York, IEEE, 1995.

- [23] Leite, J.C.S.P. and Freeman, P.A. "Requirements Validation through Viewpoint Resolution," *IEEE Trans. on Software Eng.*, 1991, 17 (12), 1253-69.
- [24] Miller, G.A., "WordNet: A Lexical database for English" *Comm. ACM*, November 1995, pp. 39-41.
- [25] Mineau, Guy W., Moulin, Bernard, and Sowa, John F., eds., *Conceptual Graphs for Knowledge Representation*, Lecture Notes on Artificial Intelligence LNAI 699, Springer Verlag, 1993.
- [26] XXXX Tim Nagle and Jan Nagle and Laurie Gerholz and Peter Eklund, eds, *Conceptual Structures: Current Research and Practice*, Ellis Horwood, 1992.
- [27] Hans Nissen, Manfred Jeusfeld, Matthias Jarke, Georg Zemanek and Harald Huber, "Managing Multiple Requirements Perspectives with Metamodels," *IEEE Software*, 12(6), pp. 37-48, 1996.
- [28] Nuseibeh, B., Kramer, J. and Finkelstein, A. "A Framework for Expressing the Relationships between Multiple Views in Requirements Specifications," *IEEE Trans. on Software Eng.*, 20 (10), 760-73, 1994.
- [29] Pfeiffer, Heather D., and Nagle, Timothy E., eds., *Conceptual Structures: Theory and Implementation*, Lecture Notes on Artificial Intelligence LNAI 754, Springer Verlag, 1992.
- [30] Polovina, Simon and Heaton, John, "An Introduction to Conceptual Graphs," *AI Expert*, pp. 36-43, 1992.
- [31] Puder, A., Markwitz, S., and Gudermann, F., "Service Trading Using Conceptual Structures," in Ellis, Gerard, Levinson, Robert, Rich, William, and Sowa, John F., eds., *Conceptual Structures: Applications, Implementations and Theory*, Lecture Notes on Artificial Intelligence LNAI 954, Springer Verlag, 1995, pp. 59-73.
- [32] Rumbaugh, James, et al., *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.
- [33] Smith, M., *Software Prototyping*, New York, New York: McGraw Hill, 1991.
- [34] Sowa, John F., *Information Processing in Mind and Machine*, Addison-Wesley Publ., Reading, MA, 1984.
- [35] Sowa, J. F. "Conceptual Graphs Summary," in *Conceptual Structures: Current Research and Practice*, Tim Nagle, Jan Nagle, Laurie Gerholz and Peter Eklund, eds., Ellis Horwood, 1992, pp. 3-52.
- [36] Tepfenhart, William M., Dick, Judith P., and Sowa, John F., eds., *Conceptual Structures: Current Practices*, Lecture Notes on Artificial Intelligence LNAI 835, Springer Verlag, 1994.
- [37] Wolf, Randy P. and Delugach, Harry S., "Knowledge Acquisition via the Integration of Repertory Grids and Conceptual Graphs," Supplemental Proceedings, ICCS '96, 4th Intl. Conf. on Conceptual Structures, P.W. Eklund, G. Ellis and G. Mann, eds., University of New South Wales, Sydney, Australia, Aug. 19-23, 1996.