Dependency Analysis Using Conceptual Graphs

Lisa Cox, Dr. Harry S. Delugach

Computer Science Dept.
University of Alabama Huntsville
Huntsville, AL 35899
lcox@cs.uah.edu, delugach@cs.uah.edu

Dr. David Skipper

Bevilacqua Research Corp.
Huntsville, AL 35815
DavidS@brc2.com

Abstract. Analysis of dependencies between entities is an important part of modeling. Whether the modeling domain is at the enterprise level or at the system or software component level, characterization, representation, and analysis of these dependencies is essential to correctly modeling the domain. For example, it is important to identify and characterize dependencies between both system and software components when trying to determine the extent of and impact of a breach in computer system security or of a malfunction in a component. Analysis of such dependencies is also greatly beneficial in both the requirements and maintenance phases of software engineering. What is needed is a formal characterization of the concept of dependency along with a more formal and unified approach to dependency analysis . This paper introduces the notion of dependency at a general level. In the present literature, an actual definition and characterization of a dependency is usually avoided, and it is difficult to separate the discussion of the dependency from the particular domain of interest. Most of the literature available implies that it is simply "understood" that a dependency can be represented by a directed arc on a graph where the dependent components are the nodes of the graph. Much work in the current literature addresses dependencies in widely varying ways. This paper attempts to formalize both the definition and characterization of a dependency in a unified approach, and then illustrates how dependencies themselves and the effect of those dependencies upon a system can be efficiently modeled using Conceptual Graphs.

1.0 Introduction

In modeling, it is usually important to identify, characterize, and understand the impact of the dependencies that exist between the entities in the model. This is

vital at all levels of modeling and in all domains. The concepts and approach presented in this paper are applicable at all levels and in each domain. We start by considering that there are many notations in use for the specification of and analysis of models. While some of these notations allow explicit specification of dependencies, some include dependency only by implication. For example, UML represents a simple dependency as a dotted arrow between components. [1]

In the software engineering domain, OSD (Open Software Description) is presently being pushed by Microsoft as a standard for describing and packaging software [5]. While OSD and the literature surrounding it are in agreement that dependency representation is important, OSD simply represents dependencies by supplying a list of other components that are required to be present before a particular software component can be installed on a system. [6]

Work has also been done in the natural language processing area dealing with dependency analysis between words of a sentence, and specific linguistic dependency types have been identified, such as the dependency between a noun and a determiner or the dependency between noun and verb. [16], [3] However, these dependencies once again are limited to the particular domain in question (i.e., linguistic dependency) and explicit definitions of an abstract dependency are not considered.

Much of the present literature takes the definition of dependency for granted and where definitions are occasionally given, they vary widely. Some sources maintain that dependencies are simply first-order logic formulae, or in database terminology, constraints [18], [4], [5]. Others insist that higher-order logic is required to express dependencies. [14] Some take a probabilistic approach and express dependencies as conditional probabilities between specified variables or look solely at dependencies from a statistical viewpoint. [10], [17], [2]. Some sources take the approach that a dependency is best modeled by the client/server relationship, and then develop the definition of dependency in client/server terms [15], [19], [8], while others specify types of dependency such as structural and functional dependencies [8] or data and value dependencies. [13].

Keller, Blumenthal, and Kar in [8] attempt a more in-depth characterization of dependencies and define six different “dimensions” of dependency, and Prost in [14] also takes a “type-based” approach to dependency analysis. However, some of the dimensions given in [8] for analyzing dependencies are actually attributes of the computer system under analysis. Once again, there is no clear delineation between the dependency itself, and the domain in which the dependency exists.

Mineau in [12] discusses the addition of functions to and the treatment of functional dependencies in Conceptual Graphs, but even Mineau does not address the explicit definition of a dependency.

This paper presents an approach for formally defining and characterizing dependencies using Conceptual Graphs. It is our contention that our approach to the definition of dependency and the use of Conceptual Graphs as a dependency language allows for a much more coherent and complete description of dependencies at the general level and explicitly delineates the characteristics of

the dependency from any domain limitations. We also expect the use of Conceptual Graphs to allow more powerful analysis of the dependencies of a given system.

2.0 Definitions

Our perspective comes from the Realist's view as defined by Hayes in [7]. We assume "a set can be a set of anything" and that "the universe can be physical or abstract or any mixture" in order to make our universe as general as possible.[7] Based upon this perspective, we then refer to an entity as anything that can be a member of such a set, and therefore can be anything we want to model. This can be an object, a concept, an organization, or any other thing to be modeled. We also make the assumption that the entities are not static. The entities can change. At this point, we simply assume the existence of something called change that happens to entities, but we deliberately do not yet attempt to define change in order that it, too, may be allowed to be as general as possible. We understand that an entity may change for at least several and possibly many reasons. The entity may have change as part of its very nature (for example, try to model a 2-year-old child without allowing for change). The entity may also be influenced to change by something outside itself. This latter type of change is of specific interest to us and it is upon this that we base our understanding of dependency. From this understanding, we assume that there are cases where the "something outside itself" possibly or potentially influences the entity to change.

We ask the reader to accept our general definitions for entity, change, and potential for change in the interest of concentrating upon dependency. We also assume the existence of a relation R between some number of entities, expressed by R(A, B, C, D, . . .) where it can be said that the R relationship exists between the entities A, B, C, D, etc.

In the general case, we define a dependency as such a relation, D, between some number of entities wherein a change to one of the entities implies a potential change to the others. We can therefore express such a general dependency as D(A, B, C, D, . . .) where D R. This general form of a dependency is shown in Figure 1. In order to emphasize the complexity of this most general type of dependency (which may exist between many entities), we refer to it as symbiosis.

As an example of this most general type of dependency, or symbiosis, we can consider the relationship between the departments within a corporation. It is easy to see that the engineering, accounting, contracts, marketing, and facilities departments are dependent upon each other. However, it is not at all easy to specify and quantify the extent of such a dependency.

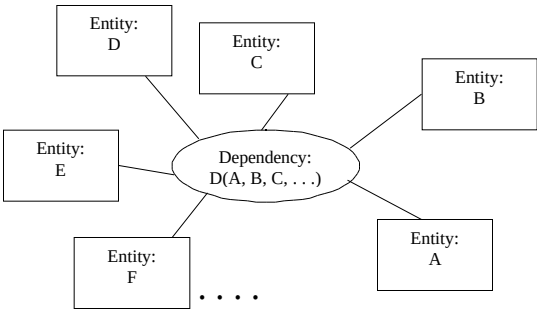

Fig. 1. Graphical representation of most general form of a dependency

As a first step in our analysis, we focus upon a much simpler type of dependency, the case of a dependency between only two entities, D(A, B). In the case where A depends upon B and B depends upon A, this dependency can be seen as a bi-directional relationship. We call this bi-directional dependency an interdependency. Given such an interdependency between two entities, we can now separate the dependency D(A,B) into at least two one-way, or unidirectional dependencies d1(A, B) and d2(B, A). We can be sure that this is always the case, because we have included "independent" in our type hierarchy for dependencies (refer to Figure 4).

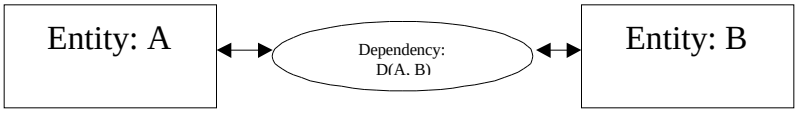

Fig. 2. Bi-directional dependency, or interdependency, between two entities

In the simplest case of a dependency, a unidirectional dependency between two entities, d(A, B), we can say that A depends upon B. If A depends upon B, then a change in B implies a potential or possible change in A. As in Keller, et. al.[12], we refer to A as the dependent and B as the antecedent. This definition of the simplest form of a dependency is very like the definition of dependency given in [1] and is depicted in Figure 3.

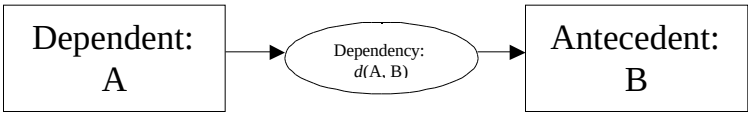

Fig. 3. Graphical representation of the simplest dependency

Again, it is important to note that this definition of the simplest case of dependency expresses a one-way direction for the dependency. As Briand, Wust, and Lounis [2] point out, it is not only possible, but common that a bi-directional

dependency exists; and, given the definition of the most general form of dependency above, it is also conceivable to have such an interdependency demonstrated between N entities where N>2.

Our initial work is based upon the decomposition of complex dependencies into unidirectional, binary relations. The complex dependency can be broken into some number of unidirectional dependencies. As described above, it is easy to see that in the case of an interdependency between two entities, the bi-directional dependency can be described using at least two one-way dependencies between the two entities. We expect that in a case of symbiosis among N entities, the symbiosis can be represented by at least

$$2\binom{N}{2}$$

unidirectional dependencies. We use the term "at least" here because there may be multiple types of dependency existing between any two entities. For example, both an intermittent, time-based dependency and a static structural dependency may be involved in the interdependency. Even if the dependency is of a single type, such as a functional dependency, it could include several different and specific "needs" of the entities. In that case, a separate unidirectional dependency could be defined for each specific need. Our continuing research will include a more in-depth investigation of this expectation.

3.0 Describing a dependency

Now that we have defined both a general dependency and the most simple dependency, we need to discover the characteristics that are inherent in all dependencies and we need to investigate the types of dependency that are possible. Our research is focusing on the very ambitious attempt to produce what might be called an ontology of dependencies. This includes both the identification of a set of attributes which apply to every dependency and the development of a general dependency type hierarchy based upon those attributes.

3.1 Attributes which describe a dependency

Keller, et. al.[8] is the only source in which we have found an attempt at the classification of dependencies based upon such attributes. Keller, et. al.[8] lists six attributes of dependency which are represented as orthogonal axes in a six-dimensional dependency space wherein each dependency can be graphed. Our initial set of attributes, which are applicable to all dependencies, includes two attributes from [8], criticality and strength. However we believe that the other four attributes cited by [8], rather than being associated with the dependency, would be more properly represented as attributes associated with the system components (the entities A and B) or with the system, itself. For example, the "component type" cited by [8], is not an attribute of a dependency as much as it is

an attribute of the entity, A, being modeled, and the attribute "dependency formalization" is actually dependent upon the particular system in question.

To the two attributes we have taken from [8], we have added the attributes of impact, sensitivity, stability, and need as important to all dependencies. Keller et. al. [8] also addresses the issue of "time", although it is not included in the six-dimensional dependency space. This is very like the attribute we have named stability. The following represent our current definitions of our initial set of attributes:

> **Sensitivity** (or fragility) – how vulnerable to compromise or failure is this dependency? Possible values for this attribute are Fragile, Moderate, and Robust.
>
> **Stability** (like "time" in [8]) – a measure of the continuity of the dependency's vulnerability to compromise or failure (sensitivity) over time. One way of looking at stability is to ask the question: "When is the dependency fragile?" Possible values for this attribute are Extremely Stable, Infrequent, Periodic, Certain Defined Times only, etc.
>
> **Need** – what "need" of entity A is fulfilled by entity B? This can be expressed as a list of particular capabilities upon which this dependency is based. Possible values for this attribute include Authorization, Resources Provided, Testing, or at lower levels could include Text Editing, Computation, Network Access, File Save/Retrieval, etc.
>
> **Importance** (or criticality) – what is the weight of this dependency as a determinant of entity A's success, or how critical is this dependency to the goals and overall function of entity A? Possible values for this attribute are: Not Applicable, High, Medium, and Low.
>
> **Strength** – a measure of the frequency of the need or the importance of this dependency, from entity A's viewpoint. How often or how much does entity A rely upon this dependency in any particular time period? One way of looking at Strength is to ask the question: "How often does this dependency's importance or need come into play?" Possible values of this attribute are Daily, Hourly, Yearly, etc. or a numeric value representing how often the dependency is an issue during a particular time period.
>
> **Impact** – in what way is the entity's function affected by compromise or failure at this particular dependency? Possible values for this attribute are: None, Mission Compromised, Information Unreliable, Performance Degraded, Corruption/Loss of Information/Communication.

This represents our initial attempt to identify the set of attributes which are applicable to all types of dependencies. Using this initial set of attributes, we are able to determine an initial version of a hierarchy of dependency types. We

expect that if it were possible to identify a complete set of such attributes, that we should then be able to identify all possible dependency types in our hierarchy.

3.2 Dependency type hierarchy.

Once a complete set of dependency attributes is identified, it will then be possible to establish a type hierarchy, resembling a lattice, based upon those attributes and their values. Using this hierarchy, specific types of dependency are characterized and related to each other, and dependency types can be chosen to be applicable to particular domains. Eventually, it should be possible to fully populate the dependency type hierarchy based upon the attributes identified. Figure 4 contains a portion of the dependency type hierarchy identified so far. From the types shown in this structure, it is now possible to analyze the dependencies discussed by each of our sources and indicate where in the structure their particular approach to dependency lies.

Several of our sources assume no more detail about a dependency than that it is a directed arc between two entities. [3], [5], and [9] are examples.

Mineau [12] goes into specifics about functional dependencies. In Lukose and Mineau[11], data dependencies are explicitly referenced, and the messages passed between actors can also be seen as data dependencies. Lukose's "control marks" and "control maps" also represent dependencies which are reflected in our hierarchy as the control dependency type. These dependencies appear to lie near the bottom of our structure as functional, data, and control dependencies respectively.

Lukose and Mineau [11] also discuss co-reference links between concepts. A co-reference link indicates that two concepts exist which describe the same entity. We have allowed that two or more representations for the same entity may be required and have included a co-reference dependency in the structure.

Briand et. al. [2] refers to data and control dependencies in particular metrics analyzed, but also introduces the dependency between object classes based upon inheritance. That type of dependency lies in the "requires" section of the structure as can be seen in Figure 4.

4.0 Using Conceptual Graphs as a dependency language.

Given the definition of dependency above, it is now straightforward to map dependencies into conceptual graphs. First, the definition of the simplest dependency is encoded in Conceptual Graph terms. Figure 3, depicting such a dependency is already in Conceptual Graph form. From there, the graph may be relationally expanded to include the definition of the dependency using its attributes. This conceptual graph is shown in Figure 5.

The relation, "dependency" has now been expanded into a graph defining a concept of "Dependency" which is related to the previous concepts of

"Dependent" and "Antecedent" and which is also now associated with attributes characterizing the most general dependency.

Note that the conceptual graph representation allows us to easily represent the most general case and also to expand the general graph in order to represent more specific information about the dependency as it becomes available, i.e. the Conceptual Graph representation facilitates modeling at multiple levels of detail simultaneously. This addresses one of the most difficult problems in modeling, the efficient representation of and processing of entities modeled at multiple levels of fidelity. Using Conceptual Graphs, the scalability problem becomes much less difficult and in some cases is solved altogether.

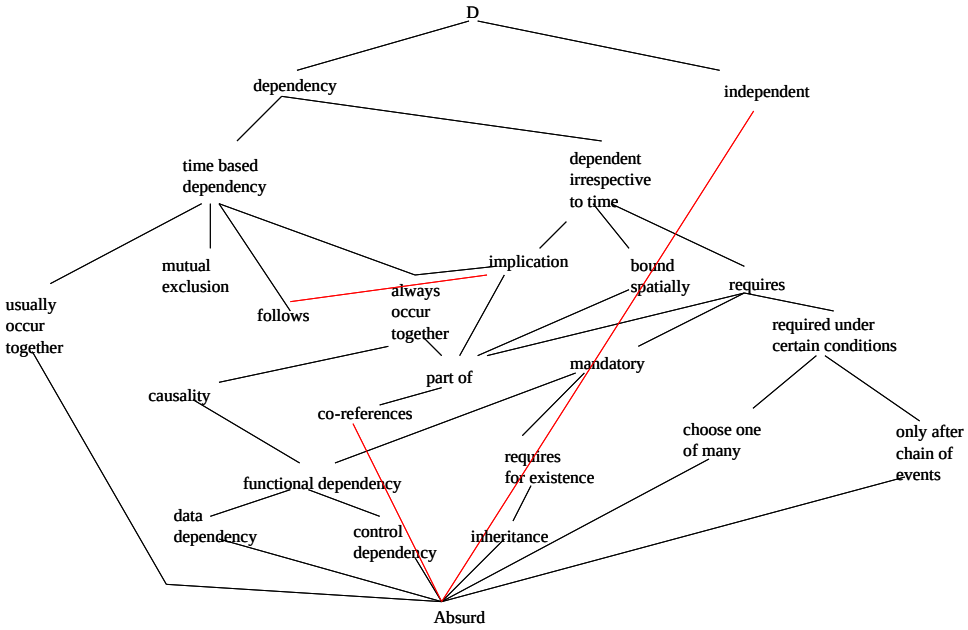

Fig. 4. Dependency type hierarchy

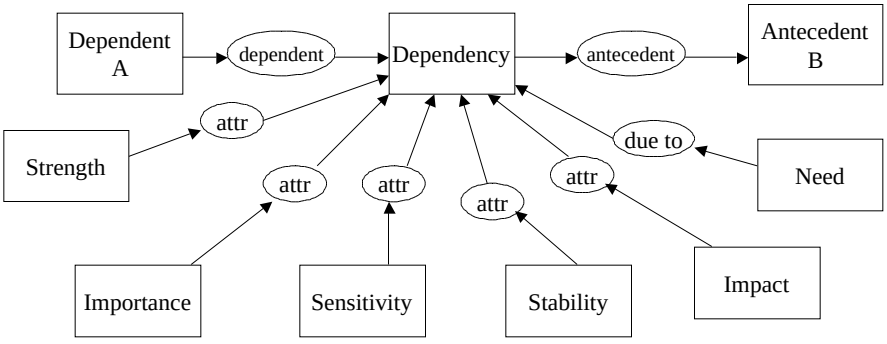

Fig. 5. Relationally expanded dependency graph

For applications such as those in [3], that need no more knowledge of the dependency than the direction of the relationship (or the directed arc), the graph shown in Figure 3 need not be expanded at all. Also note that if the parts of speech used in [3] are defined as subtypes of "Dependent" and "Antecedent", then restriction of the graph shown in Figure 3 allows the expression of all the dependencies used by that source.

As noted earlier, [8] focused upon dependencies between hardware and software system components in a distributed system. All six dependency attributes cited in [8] are vital to analysis of that domain. While our dependency attributes specifically include two attributes from [8], the issues surrounding the other four must also be addressed. In order to address the others, we first need to restrict the dependent and antecedent to a subtype of "computer system component." This representation allows the information pertaining to the system components to be put into attributes associated with those concepts in order to leave the definition of the dependency concept uncluttered. A possible definition of "computer system component" which will include the information required by [8] is shown in Figure 6.

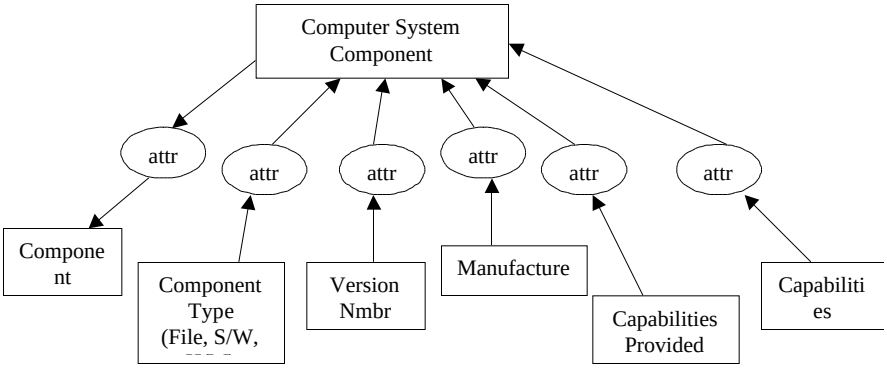

Fig. 6. Computer system component definition

In this way, the two requirements for "component type" and "component activity"[8] (which we have named "capabilities provided") are represented as attributes of the components and thereby influence the analysis, but are separated from the dependency itself.

The dimension of "locality"[8] is more difficult to deal with. But if we introduce the idea of dependency chains, as indeed were introduced in [8], then a dependency can be defined between entities A and E, d1(A, E) where the set of dependencies, {d2(A, B), d3(B, C), d4(C, E)} form such a dependency chain. This introduces a limited transitivity of dependencies: given the possibility that d2, d3, and d4 can be of different dependency types, it is very difficult to draw conclusions about the nature of such transitivity without examining the specific

definitions of the dependency types. However, if d2, d3, and d4 are identified as dependency types that are indeed transitive, then the attribute of "locality"[8] can then be implemented by counting the "hops" on the fully expanded dependency chain and including a weighting factor or "importance" such that a dependency "hop" between a software component and a hardware component is more significant than one between two software components.

We also have not addressed the last dimension given [8]. "[D]ependency formalization" does not appear in our attribute list. Keller et. al. define that particular dimension as "a metric [signifying] how expensive and/or difficult [it is] to acquire and identify this dependency," particularly relating to the "degree it can be determined automatically." Although we understand why this particular "dimension" is important given the domain of focus, we again think it is better to separate this from the attributes of the general dependency. In some systems a dependency may be extremely simple to "determine automatically" if UML descriptions of system components are available, while an identical dependency may be extremely difficult to identify "automatically" in a legacy system which has little supporting documentation.

5.0 Our approach applied to two examples

To illustrate the application of our approach, we now present two examples. In the first, we consider the analysis of a particular dependency in a computer system from the information security perspective. A complex dependency exists between network browser (Brows), e-mail package (EMail), word-processing package (WP), and the hardware/software that provides network access (Net). That dependency may be broken into at least twelve unidirectional dependencies as follows.

d1(Brows, EMail)	d2(Brows, WP)	d3(Brows, Net)
d4(EMail, Brows)	d5(EMail, WP)	d6(EMail, Net)
d7(WP, Brows)	d8(WP, EMail)	d9(WP, Net)
d10(Net, Brows)	d11(Net, EMail)	d12(Net, WP)

In systems where the network browser and the e-mail package are unrelated, d1 and d4 will be extremely weak dependencies, to the point where they are categorized as independent. In other systems, where the same software package is used to provide both these services, the dependencies will be much stronger. In the case where the word processing package, WP, depends upon the network browser, Brows, to download new fonts, the dependency d9(WP, Net) can be replaced by the dependency chain: { d7(WP, Brows), d3(Brows, Net) }.

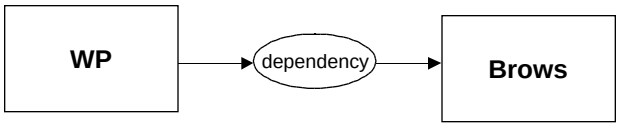

Fig. 7. Dependency Example

In this example, the attributes of each dependency could be given values where data is available, but left without a value if the attribute does not involve itself in the analysis.

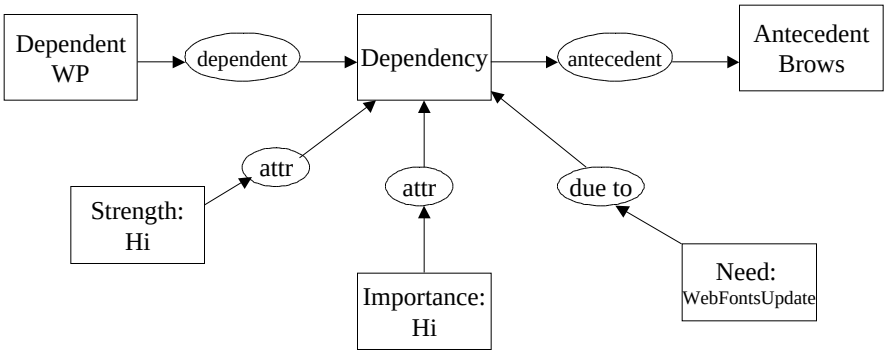

Fig. 8. Dependency Specifics

In the second example, we look at a dependency encountered when designing a model at the enterprise level. A complex dependency, or symbiosis, exists between certain entities within the enterprise, such as the contracts department, the proposal department, and the engineering department. Let us identify one particular dependency that exists at the point of decision on whether or not to bid upon a contract. The contracts and proposal departments depend upon the engineering department for technical knowledge of the scope of the effort, estimates of the cost of performing the contract, estimates of the cost of bidding upon the contract, and possibly estimates of the chances of a contract win. The engineering department and the contracts department depend upon the proposal department for resources and expertise that will allow the preparation of and delivery of a proposal. The engineering and proposal departments depend upon the contracts department for legal knowledge and possibly for a "go/no-go" decision.

This complex dependency may be broken into a set of simple dependencies that explicitly specify each one-way dependency identified. The following list represents some of the dependencies that might be identified:

d1(ContractsDept, ProposalDept)
d2(ContractsDept, EngineeringDept)
d3(ProposalDept, ContractsDept)
d4(ProposalDept, EngineeringDept)
d5(EngineeringDept, ContractsDept)
d6(EngineeringDept,ProposalDept)

We can then determine the attributes associated with each dependency. For example, we could determine that because the enterprise has a documented and often-used policy for proposal preparation, the sensitivity of both d1 and d6 is "robust." However, we also could assign a sensitivity of "fragile" for d4 and d2 if there has been some recent history wherein the engineering department has been less than helpful during proposal efforts. In that case, we could also assign an impact rating of "information unreliable" and an importance rating of medium if the enterprise is of the opinion that the contract bid can take place without detailed engineering input, but would be more confident of their efforts if that detail were available. The stability of d4 could be assigned to "3rd and 4th weeks" if there are two particular weeks when input from the engineering department to the proposal department will make a difference in the proposal department's output. Given the description above, the need associated with d4 could be formulated as a list of capabilities or of outputs from the engineering department required by the proposal department: estimate of technical scope, contract cost estimate, proposal preparation cost estimate, and estimate of win probability. It can also be seen that the modeler might choose to break d4 down further into a set of simple dependencies where the need associated with each is a single product or output.

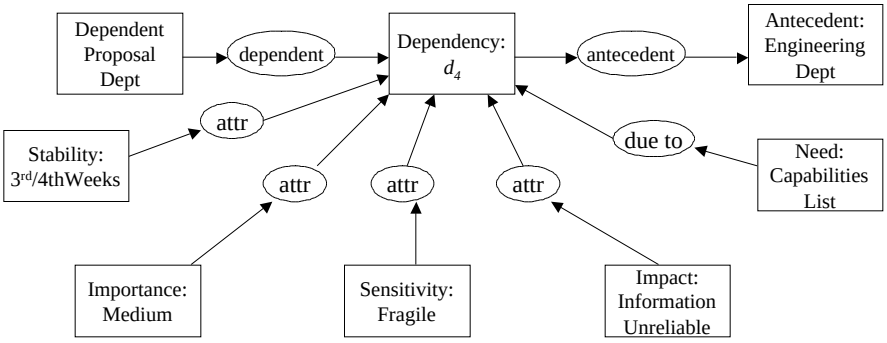

Fig. 9. Dependency Example, Enterprise Level

6.0 Significance

Without a type based approach to dependency analysis, it becomes extremely difficult to distinguish between widely differing dependencies. For example, systems that easily represent functional dependencies have difficulty dealing with causality. The statement that "a dependency exists between" two entities carries almost no information. Is the dependency a causal relationship? Is it a mutually exclusive relationship? Both relationships are dependencies and need to be analyzed in much different ways. A type based approach to dependency analysis will allow the modeling of dependencies at the traditional level where not much is

known besides direction of the dependency. But it will also allow much more in-depth analysis of complex relationships in multiple domains.

7.0 Conclusion

The approach given in this paper shows that dependencies can be grouped based upon the identification of attributes applicable to all dependencies. From that set of attributes, a dependency type hierarchy can be produced that will cover all dependencies found in the present literature. Our initial research indicates that this approach provides a more general and unified approach to dependency analysis. We have also shown that Conceptual Graphs provide a powerful approach to represent, characterize, and analyze dependencies between the entities in a model. Using Conceptual Graphs, we can more easily model entities at different levels of model fidelity and when only partial information is available. We are planning continued research which will extend these results to a broader set of practical applications.

8.0 Continuing work

Continuing research is needed to fully investigate and populate the dependency hierarchy in order that all relevant dependency types can be investigated in the approach. We expect that new dependency types will be easily incorporated into our approach simply because it is a type based approach. Also, as we discover more attributes which are pertinent to the general case of dependency, they can be easily added into the existing structure. A more complicated issue which needs to be addressed is the representation and analysis of the most general form of a dependency. A method is needed for decomposing such dependencies in order to simplify analysis. In addition, research is needed in the formal analysis of the method, and in analysis of the complexity of that method. We also need to investigate a more formal definition of our assumptions for entity, change, and potential for change, upon which our definitions are based.

Acknowledgment

This work has been funded in part by the Air Force Research Laboratory, Wright Research Site, for the Defense Advanced Research Agency (DARPA) Autonomic Information Assurance Program.

References

[1] G. Booch, I. Jacobson, J. Rumbaugh, and J. Rumbaugh, The Unified Modeling Language User Guide: Addison-Wesley, 1998.

[2] L. C. Briand, J. Wust, and H. Lounis, "Using Coupling Measurement for Impact Analysis in Object Oriented Systems," presented at IEEE International Conference on Software Maintenance (ICSM98), Bethesda, MD, 1998.

[3] C. Chelba, D. Engle, F. Jelinek, V. Jimenez, S. Khudanpur, L. Mangu, H. Printz, E. Ristad, R. Rosenfeld, A. Stolcke, and D. Wu, "Dependency language modeling, 1996 Large Vocabulary Continuous Speech Recognition Summer Research Workshop," Center for Language and Speech Processing, Johns Hopkins University Technical Reports, Research Note 24, April 15, 1997.

[4] V. M. Crestana-Jensen and A. J. Lee, "Consistent Schema Version Removal: An Optimization Technique for Object Oriented Views," IEEE Transactions on Knowledge and Data Engineering, vol. 12, pp. 261-280, 2000.

[5] R. S. Hall, D. Heimbigner, and A. L. Wolf, "Software deployment languages and schema," Dept. of Computer Science, University of Colorado CU-SERL-203-97, December 18, 1997.

[6] W. J. Hansen, "Deployment Descriptions in a World of COTS and Open Source," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 1999.

[7] P. Hayes, "Aristotelian and Platonic Views of Knowledge Representations," presented at Second International Conference on Conceptual Structures (ICCS94), College Park, MD, 1994.

[8] A. Keller, U. Blumenthal, and G. Kar, "Classification and Computation of Dependencies for Distributed Management," Proceedings of the Fifth International Conference on Computers and Communications (ISCC 2000), 2000.

[9] A. A. Kountouris and C. Wolinski, "High Level Pre-Synthesis Optimization Steps Using Hierarchical Conditional Dependency Graphs," presented at 25th Euromicro Conference (EUROMICRO '99), Milan, Italy, 1999.

[10] R. Levinson and A. R. Goodwin, "Explorations in Scientific Thinking: a Systems Theoretic Approach: Chapter 2," in Scientific Thinking: a Systems Theoretic Approach. Santa Cruz, CA, 2000, p. 65.

[11] D. Lukose and G. W. Mineau, "A Comparative Study of Dynamic Conceptual Graphs," Brightware Inc/Department of Computer Science, Université Laval,, New York, NY/Quebec City 1998.

[12] G. W. Mineau, "Views, Mappings, and Functions: Exxential Definitions to the Conceptual Graph Theory," presented at Second International Conference on Conceptual Structures (ICCS94), College Park, MD, 1994.

[13] M. Papazoglou, A. Delis, A. Bouguettaya, and M. Haghjoo, "Class Library Support for Workflow Environments and Applications," IEEE TRANSACTIONS ON COMPUTERS, vol. 46, pp. 673-686, 1997.

[14] F. Prost, "A Static Calculus of Dependencies for the l-Cube," presented at 15 th Annual IEEE Symposium on Logic in Computer Science (LICS'00), Santa Barbara, CA, 2000.

[15] J. Rumbaugh, I. Jacobson, and G. Booch, The Unified Modeling Language Reference Manual: Addison-Wesley, 1998.

[16] D. D. K. Sleator and D. Temperley, "Parsing English with a Link Grammar," School of Computer Science Carnegie Mellon University, Pitsburg, PA CMU-CS-91-196, October 1991.

[17] V. S. Subrahamanian, P. Bonatti, J. Dix, T. Eiter, and F. Ozcan, Heterogeneous Agent Systems, 1st ed. Cambridge, Mass: MIT Press, 2000.

[18] B. Thalheim, Entity-Relationship Modeling: Foundations of Database Technology. New York: Springer-Verlag, 1998.

[19] E. S. K. Yu, J. Mylopoulos, and Y. Lespérance, "AI Models for Business Process Reengineering," IEEE Expert, vol. 11, pp. 16-23, 1996.