Proceedings of the 5th National Conference; INDIACom-2011
Computing For Nation Development, March 10 – 11, 2011
Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi

# Conceptual Dependency

**Gurvinder Singh[1], Ravleen Kohli[2], Dilip Mujalde[3], Piyush Pradhan[4]** and **Priyanka Mehra[5]**
[1,3,4,5]CIIT, Indore, [2]SRGPI, Indore
[1]gurvinderengg@yahoomail.com

**ABSTRACT**
*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. In the software engineering domain, OSD (Open Software Description) is presently being pushed by Microsoft as a standard for describing and packaging software. 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. 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. 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.*

**KEYWORDS**
Dependency, Attribute, Entity

## INTRODUCTION

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. Others insist that higher-order logic is required to express dependencies. Some take a probabilistic approach and express dependencies as conditional probabilities between specified variables or look solely at dependencies from a statistical viewpoint. 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, while others specify types of dependency such as structural and functional dependencies or data and value dependencies. Keller, Blumenthal, and Kar attempt a more in-depth characterization of dependencies and define six different "dimensions" of dependency, and Prost also takes a "type-based" approach to dependency analysis. However, some of the dimensions given in for analyzing dependencies are actually attributes of the computer system under analysis. Once again, there is no clear delineation between the dependencies itself, and the domain in which the dependency exists. Mineau 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. 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. Our perspective comes from the Realist's view as defined by Hayes. 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. 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

 ISSN 0973-7529 ISBN 978-93-80544-00-7

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.

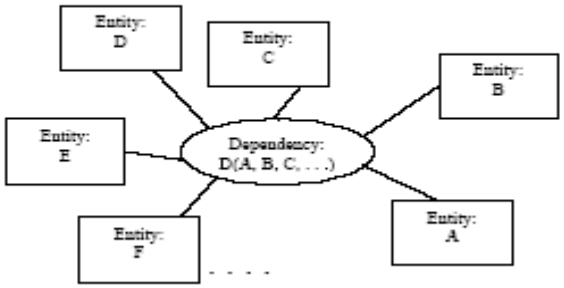


*Figure1Graphical representation of most general form of a dependency*

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 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).


*Figure2Bi-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.


*Figure 3Graphical 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. 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 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 (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.

**MODEL**

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 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.

**Attributes that describe a dependency**

There are 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, criticality and strength. However we believe that the other four attributes cited by , rather than being

 ISSN 0973-7529 ISBN 978-93-80544-00-7

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 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, we have added the attributes of impact, sensitivity, stability, and need as important to all dependencies. 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 is the list of attributes of the dependencies:

**Attributes of Dependency**

**1. Sensitivity** (or fragility) – how vulnerable to compromise or failure is this dependency? Possible values for this attribute are Fragile, Moderate, and Robust.

**2. Stability** (like "time") – 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, and Certain Defined Times only, etc.

**3. 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.

**4. 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.

**5. 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.

**6. 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.

**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 indicates 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.

**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.

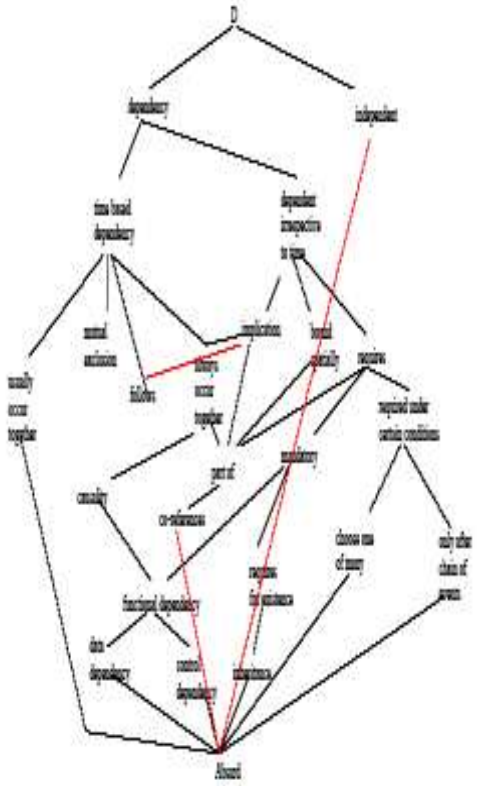

*Figure 4Dependency type hierarchy*

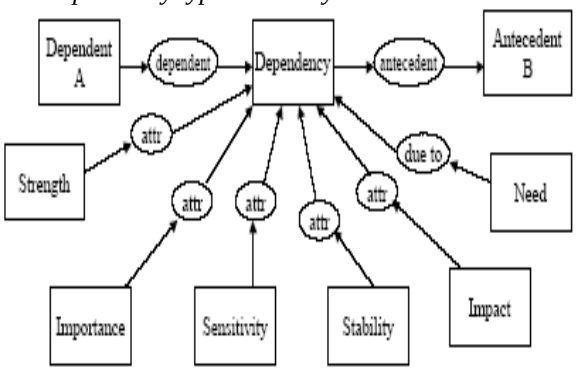


*Figure 5 Relationally expanded dependency graph*

For applications such as those in, 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 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, 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.

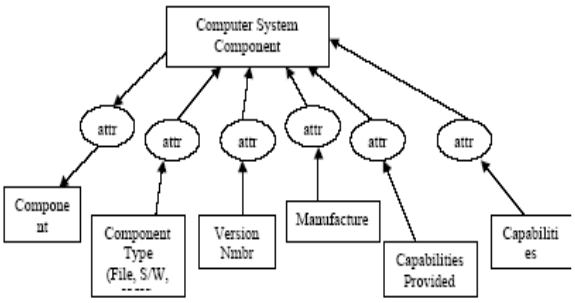


**Fig. 6.** Computer system component definition

In this way, the two requirements for "component type" and "component

activity" (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" 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. Dependency formalization" does not appear in our attribute list. Keller et. al. define that particular dimension as "a metric [signifying] how expensive and/or difficult 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.

 ISSN 0973-7529 ISBN 978-93-80544-00-7

**RELATED WORK**

[1] describe a methodology that can integrate n database views simultaneously. The methodology consists of transforming a database view into an intermediate representation, based on the conceptual dependency theory. The conceptual representations corresponding to the views are then combined to form a "global" representation, which is subsequently converted back to a data model that represents the global, integrated schema. Our methodology makes use of the semantic content of a database view in the integration process, unlike other view integration methodologies proposed in the literature. We show, by examples, how this approach can eliminate multiple restructuring of constituent views in the integration process

**REFERENCES**


[1]. Adam, N.R.; Gangopadhyay, A.;An N-ary view integration method using conceptual dependencies System Sciences, 1995. Proceedings of the Twenty-Eighth Hawaii International Conference on Volume 3, 3-6 Jan. 1995 Page(s):391 - 397 vol.3

[2]. Moore, L.E.; O'Neal, M.B.;A semantic interpreter for a transportable command language interface Applied Computing, 1990., Proceedings of the 1990 Symposium on 5-6 April 1990 Page(s):202 – 208

[3]. Nurcan, S.; Claudepierre, B.; Gmati, I.;Conceptual dependencies between two connected IT domains: Business/IS alignment and IT governance Research Challenges in Information Science, 2008. RCIS 2008. Second International Conference on 3-6 June 2008 Page(s):87 – 98

[4]. Chang, S.K.; Orefice, S.; Polese, G.; Baker, B.R.; Deriving the meaning of iconic sentences for augmentative communication Visual Languages, 1993., Proceedings 1993 IEEE Symposium on 24-27 Aug. 1993 Page(s):267 – 274

[5]. Aramaki, S.; Nagai, T.; Kawamura, M.; Yayoshi, K.; Hatada, Y.; Tsuruoka, T.;A Robot Programming Based on Frame Representation of Knowledge Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on 16-19 Oct. 2007 Page(s):903 – 908

[6]. Aramaki, S.; Nagai, T.; Kawamura, M.; Hatada, Y.; Tsuruoka, T.;Human-Robot Interface by using frame like knowledge base Automation Science and Engineering, 2007. CASE 2007. IEEE International Conference on 22-25 Sept. 2007 Page(s):729 – 734

[7]. Aramaki, S.; Nagai, T.; Yayoshi, K.; Tsuruoka, T.; Kawamura, M.; Kurono, S The Generation method of A Robot Task Program for multi-modal Human-Robot Interface.; Industrial Technology, 2006. ICIT 2006. IEEE International Conference on 15-17 Dec. 2006 Page(s):1109 – 1114

[8]. Kunrong Chen; Rajich, V.;RIPPLES: tool for change in legacy software Software Maintenance, 2001. Proceedings. IEEE International Conference on 7-9 Nov. 2001 Page(s):230 -239

[9]. A knowledge representation for the communication between robots Morita, T.; Aramaki, S.; Kurono, S.; Kagekawa, K.; Robot and Human Communication, 1993. Proceedings., 2nd IEEE International Workshop on 3-5 Nov. 1993 Page(s):308 -313

[10]. .Acquiring and managing knowledge using a conceptual structures approach: introduction and framework Berg-Cross, G.; Price, M.E.; Systems, Man and Cybernetics, IEEE Transactions on Volume 19, Issue 3, May-June 1989 Page(s):513 – 527

[11]. PHRAN-SPAN: A Natural Language Interface for System Specifications Granacki, J.; Parker, A.C.; Design Automation, 1987. 24th Conference on 28-1 June 1987 Page(s):416 – 422

[12]. The role of object-oriented techniques and multi-agents in story understanding Chavez, N.R., Jr.; Hartley, R.T.; Integration of Knowledge Intensive Multi-Agent Systems, 2005. International Conference on April 18-21, 2005 Page(s):580 - 585