The Problem of Conceptual Incompatibility
Exploring the Potential of Conceptual Data Independence to Ease Data Integration

Simon McGinnes

Trinity College Dublin, Dublin 2, Ireland
simon.mcginnes@tcd.ie

Abstract. Application interoperability and data exchange are desirable goals, but conventional system design practices make these goals difficult to achieve, since they create heterogeneous, incompatible conceptual structures. This conceptual incompatibility increases system development, maintenance and integration workloads unacceptably. Conceptual data independence (CDI) is proposed as a way of overcoming these problems. Under CDI, data is stored and exchanged in a form which is invariant with respect to conceptual structures; data corresponding to multiple schemas can co-exist within the same application without loss of integrity. The use of CDI to create domain-independent applications could reduce development and maintenance workloads and has potential implications for data exchange. Datasets can be merged without effort if stored in a conceptually-independent manner, provided that each implements common concepts. A suitable set of shared basic-level archetypal categories is outlined which can be implemented in domain-independent applications, avoiding the need for agreement about, and implementation of, complex ontologies.

Keywords: Data integration, domain-independent design, conceptual data independence, archetypal categories.

1 Introduction: The Problem of Conceptual Incompatibility

The present massive proliferation of databases, web pages and other information resources presents a data integration problem. There is a need to use data in a joined-up way, but mechanisms are lacking that allow easy data integration in the general case; it is often hard to combine data resources. A prime reason for this is that different datasets typically have incompatible conceptual structures. Common practice in information systems (IS) design leads each organisation or software vendor to create its own idiosyncratic data structures that are incompatible with those created by others; commercial pressures can have the same effect. Standard conceptual structures are normally used only in limited circumstances, when imposed by enterprise software platforms, legislative requirements or other external constraints. In the rush to create ever-more comprehensive and powerful IS, the increasing problem of heterogeneous, incompatible conceptual structures has been left for future technology to solve.

1.1 Why Do Current Methods of Integration Not Solve the Problem?

Developers have historically faced two issues with regard to integration of systems that have distinct conceptual data structures: physical incompatibility and conceptual incompatibility. Thankfully, many technologies now exist to resolve the first issue, by physically interconnecting heterogeneous platforms; these include RPC, CORBA and web services. Programs can also be linked simply by exchanging files using a common format such as XML. However, progress on physical compatibility has exposed the deeper second issue of conceptual or semantic compatibility: the problem of reconciling implicit conceptual models.

If we wish to use several data resources in an integrated way, they must share both a common vocabulary and a common conceptual framework. This fundamental and unavoidable principle of semiotics [1] may be understood by analogy to human communication: if two people wish to exchange information effectively they must speak the same language, but they must also possess shared concepts. Conceptual compatibility thus runs deeper than mere language; for people to communicate they must interpret words identically, or nearly so, and there is no guarantee that this will be the case. Meaning is essentially personal and subjective, affected by context, culture, and so on.

Getting two programs to exchange data involves a similar problem. A common, recognisable vocabulary must be used by both sides, and the two programs must also have been programmed with common concepts, so that they can act on the data appropriately. Computers cannot yet understand data in the sense that a human does, but they can be programmed to deal sensibly with specific items of data provided that the data is of a known type; this is what we mean when we say that a program "understands" a particular concept. In practice, however, most IS share neither vocabulary nor concepts. It would be surprising if they did, given they ways they are developed and the rarity with which standard conceptual structures are applied. For this reason, linking real-world IS that have heterogeneous conceptual schemas is rarely a simple matter.

In trivial cases it can seem straightforward to map the conceptual models of distinct systems to one another. For example, two programs which manage data about customers might well use similar data structures and common terms such as *name, address* and *phone number*. But semantic complexity lurks even in apparently straightforward situations. Is a customer a person or an organisation? Are business prospects regarded as customers, or must we already be trading with someone for them to be considered a customer? What about ex-customers? Many such questions can be asked, highlighting the inconvenient truth that most concepts are more complex than they seem, when one scratches the surface, and certainly far more complicated and esoteric than the trivial example quoted above. Uniting separately-developed conceptual structures can be challenging even for expert developers working with systems in closely-related domains [2]; it can be difficult to discern what data structures are intended to signify and what unstated assumptions have been made.

Another approach to data integration involves the use of automated schema matching, and tools for this purpose have been developed with some success [3]. But there is an inherent limit to the ability of automated matching strategies to operate reliably in the general case. Software cannot easily call upon context, domain expertise and general knowledge to understand and disambiguate the meaning of conceptual structures [4]. Again, the analogy of human understanding is relevant. When conversing with others, we draw upon our prior knowledge to understand what is meant. A person without prior knowledge has no hope of understanding what somebody else says. This analogy suggests that automated schema matching strategies must first overcome the grand challenge of accumulating and applying general knowledge before they can be expected to extract the semantics in arbitrary schemas with sufficient reliability [5].

In summary, semantic issues make it difficult, as a rule, to match conceptual models between IS—especially since most IS have idiosyncratic designs and complex conceptual structures that are based on unstated assumptions [6]. Conceptual incompatibility therefore presents a major barrier when we attempt to link IS. And this ignores the scalability problem, that integrating systems typically requires a good deal of interface code which must be crafted, onerously, by hand.

Conceptual incompatibility is also a problem for end users [7]. It means that we must adopt a different mental model of reality each time we use a different program. For example, consider how the concept *person* is treated in different software products from the same vendor. In Microsoft Word, people are represented merely as "users". In Microsoft Project, people are considered from a management perspective as "resources". In Microsoft Outlook people are considered as "contacts". Although these different treatments refer to the same underlying entity (a person), they are in fact three quite distinct mental concepts, each with its own meaning and implications.

The same applies to most software applications: each application takes a unique perspective on reality to suit its own purpose. The user is left to mentally reconcile the various perspectives. This is at best confusing, since the concepts may be overlapping or orthogonal, and applications rarely spell out precisely what they mean by any given term. It can also be a problem for developers, who often lack understanding of the domain concepts in applications [8].

2 Standardisation of Conceptual Structures

The reliance on post-hoc system integration implicitly facilitates the trend towards growth in conceptual incompatibility. By allowing heterogeneous applications to proliferate, we are effectively supporting the development of incompatible conceptual structures. This is a major concern [9]; "the Semantic Web should not sit on the Tower of Babel" [10]. Some means is needed of limiting heterogeneity or at least of facilitating the job of reconciling heterogeneous conceptual structures.

An alternative to the idea of reconciling data structures is to design IS such that they conform *ab initio* to standard conceptual structures. The use of standardised conceptual structures could have benefits for a software industry which is experiencing uncontrolled growth in conceptual incompatibility and its associated costs. This is an idea with some support, and many competing standards, formats and ontologies have been developed over the years for use in different application domains.

Parallels can be drawn with the development of other industries. For example, in the early railway industry, locomotives and track were crafted individually, resulting in a variety of incompatible gauges and coupling mechanisms [11]. At first, the absence of standards was unimportant, because railways were not linked. But when integration of the network became important, the existing ad hoc design practices soon became a barrier to progress. Standards were needed, addressing not just infrastructure but also more fundamental concepts such as *time* [12]. Competing standards faced resistance and controversy. For example, broad gauge was regarded as technically superior, but lost out to standard gauge in some regions after decades of competition.

Table 1 lists other spheres in which integration has led to the need for standards, often despite conflict and opposition. In all of these domains, growth led to increasing interconnection and this in turn created a need for standardisation. In retrospect, the inevitability of such standards is obvious, given the need for interoperability, and the alternative is unthinkable. Nevertheless, the adoption of standards is often painful because it requires that some or all participants give up their own solutions. We argue that the software industry has yet to fully confront this issue with regard to conceptual structures.

A common argument against standardisation is that a single solution cannot possibly be the best technical choice for every situation. Yet many IT standards have emerged despite superior competition. SQL became the standard database query language, despite the existence of languages considered more powerful and easier to use [13]. TCP/IP is dominant despite widespread promotion of the OSI standard [14]. The QWERTY keyboard layout remains the standard despite the development of more ergonomic layouts [15].

In all of these cases, adopting standards has provided widespread benefits despite the pain involved for those with vested interests. We suggest that the IS field could obtain similar benefits by standardising conceptual structures. Implementation of standard conceptual structures could make interoperation more straightforward, perhaps even offering the ability to integrate information resources in a plug-and-play fashion. The alternative is a future of information islands, multiple interfaces, frequent schema translation operations, with attendant complexity and opportunities for conceptual confusion.

Table 1. Examples of Standards

Sphere	Examples of standards
Finance	Accounting conventions
	International payment systems
Law	Legal harmonisation within the European Union
	International double taxation treaties
Electricity	Adoption of AC with standard frequency and voltage
	Use of standard electrical connectors
Electronic media	VHS (despite alleged technical inferiority to Betamax)
	Blu-Ray

Much current thinking on data integration centres on tagging, using technologies such as the Semantic Web, RDF, linked data, ontologies and microformats [16]. The hope is that tagging will allow applications to exchange and process data without intervention. "We're not that far from the time when you can click on the web page for the meeting, and your computer, knowing that it is indeed a form of appointment, will pick up all the right information, and understand it enough to send it to all the right applications" [17].

How feasible is this? Referring to the discussion in Section 2, this kind of interoperability would require both a shared vocabulary and a shared conceptual framework. That means that each piece of data must be named in a recognisable way (vocabulary) and its name must refer to some shared meaning (concept). Organisations wishing to exchange tagged data must therefore agree on a common terminology, which they can map to their proprietary data structures, and they must also agree on common concepts, which they can code into their applications. For an application to possess a concept means that the application recognises what to do with data pertaining to that concept. Asking two software applications to exchange data in the absence of common concepts is rather pointless, since the receiving application can do little with the data except store it.

Microformats offer an illustration. They provide a common terminology (hRecipe, hCard, etc.) and also a series of common, if rather simplistic, concepts that applications can be programmed to share. The development of microformats is perhaps a pragmatic reaction against large-scale ontology development, the seemingly never-ending effort to create universal "conceptual models of everything" [18]. Microformats offer the potential for quick wins because they are intended as simple, uncontroversial conceptual model snippets. They are couched at an "everyday" level of generality and therefore easy to understand [19]. By definition, microformats ignore most of the complexity of real-life conceptual structures. In particular, they neglect the relationships between concepts, which is where most conceptual complexity lies. This is what allows developers to use microformats so readily.

But, while it is easy to envisage agreement on simple, well-known concepts such as recipes and appointments, it is in the nature of conceptual structures to quickly become complex. Efforts to create reusable, generic structures can soon result in hard-to-understand abstractions that are less useful for any particular application. Microformats remain useful while they remain simple and disconnected from one another, but when there is a need for integration to reflect the real-world relationships between concepts, the complications associated with larger-scale ontologies quickly arise [20].

In summary, it remains difficult to agree on standards for the domain-specific concepts found in much enterprise data, particularly when IS are viewed as a source of competitive advantage and best practice in IS design begins with idiosyncratic conceptual structures. Historically, previous efforts at conceptual standardisation have encountered similar problems for similar reasons [21].

2.1 Ontologies as a Potential Solution to Conceptual Incompatibility

Ontologies are a current focus of attention in conceptual standardisation. *Domain* (industry-specific) ontologies are now available or in development, each created more or less in isolation to suit the needs of a particular business area. Domain ontologies are normally incompatible with one another and lack common concepts. As a result, matching two arbitrary domain ontologies can be challenging. In contrast, *upper* ontologies are more wide-ranging; so as to encompass a range of application domains they typically include broad and generic abstractions. One approach to ontology matching makes use of this by mapping domain ontologies to one another using the high-level abstractions in upper ontologies [22].

Ontologies offer a potential source of common conceptual structures and may therefore present a solution to the problem of conceptual incompatibility. They can be used to integrate applications in two primary ways. One is by acting as a design blueprint, so that applications are constructed to share a common conceptual model. This automatically renders applications compatible provided that they do not introduce extensions or subtle variations in semantics to suit their own needs. It is therefore possible that conceptual incompatibility could be resolved, if all applications were built to conform to a single upper ontology, linked in turn to an agreed set of domain ontologies, if the ontologies in question remained relatively static. However, the task would be enormous, even if everyone could agree on a single set of ontologies to suit all purposes. Given that reality can be modelled in an infinite variety of ways, this seems unlikely. As one researcher succinctly put it, “knowledge cannot be standardised, since each day more sprouts” [23]. Others have observed that it might be more practical to have a flexible means of interpreting concepts at runtime rather than a conceptual language that is rigidly defined a priori.

The other way in which ontologies can be used to integrate applications is for each application to use its own conceptual structure or ontology, as at present, but to match up the distinct ontologies, so allowing translation and exchange of data. This is in effect the commonly-used approach. However, it seems unlikely that this approach can provide a lasting solution to the problem of integration on a large scale. It does not address the fundamental problem of conceptual fragmentation; as in schema matching, ontology matching is labour-intensive and fully-automated matching is currently infeasible in the general case.

3 Conceptual Data Independence

Below we propose an alternative solution to the problem of conceptual incompatibility. Our solution is based on conceptual data independence (CDI), which refers to the storage of data in a format that is invariant with respect to conceptual structures. A primary benefit of CDI is that it reduces the knock-on effects of changes to conceptual structures, so that development and maintenance costs can be reduced. However, CDI also offers the prospect of easier data integration. Below we give a brief explanation of CDI and how it can be achieved, and then discuss how it can assist in the data integration task. The scheme outlined below is not presented as the only or best way of implementing CDI, but as an example for illustrative purposes. We hope that it will stimulate discussion on alternative ways of achieving CDI and their respective advantages.

An aim of CDI is to avoid the need to modify applications whenever underlying conceptual structures change. This suggests that applications and databases should be designed using software structures which are independent of conceptual structures. For example, to store data about customers, one would have to construct a database structure without referring to the concept *customer*, or anything like it. This requirement contradicts current design practice, since one would normally expect to store data about customers in a "Customers" table or equivalent.

A step in the right direction is to find some invariant aspect of customers to use as a data structuring mechanism. The idea of a role is helpful here. If customers are people, then the concept *customer* is a role that people play. Roles are, by definition, transient and overlapping—we play them from time to time. The idea of a person is also a concept, but a less volatile and more universal one. Accordingly, it may help to base our data structure on the concept *person* rather than the role *customer* [24].

In general, mental concepts may be divided into roles and non-roles. Non-roles can be recorded as invariant knowledge whilst roles may be better recorded as variant or volatile data. This idea is represented in the conceptual structures shown in Figure 1. The first structure shows concepts *customer* and *supplier*. In the second, substitution of these concepts with more general ones (*person* and *organisation*) transforms the role into a relationship.

The distinction between variant and invariant knowledge is not a very rigorous one. However, there can be practical value in distinguishing concepts, which are relatively permanent, from roles, which are relatively impermanent. For example, without negotiation there is unlikely to be universal agreement on what a customer is and how a customer is defined. But it is possible to assume agreement that people exist, and this agreement is all that is needed to allow the most basic level of data exchange. Once again, the analogy of human communication is helpful; two individuals can converse effectively if they can safely assume that common basic-level concepts are shared (such as the idea of a person or a place) even if they have slightly different ideas about how these things might be defined in detail.

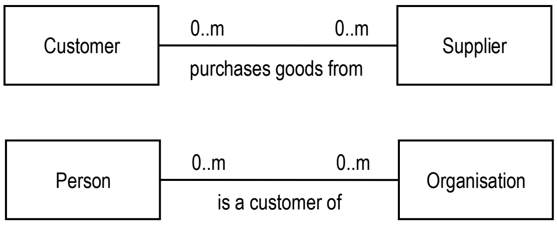

Fig. 1. Conceptual structure representing a role as a relationship

A more complex example follows. Consider a software application that handles information about product types, suppliers, stores, customers and the purchases that customers make. In a classically-designed database, the process of normalisation would lead to a separate table representing each entity type. A possible solution is illustrated in Figure 2.

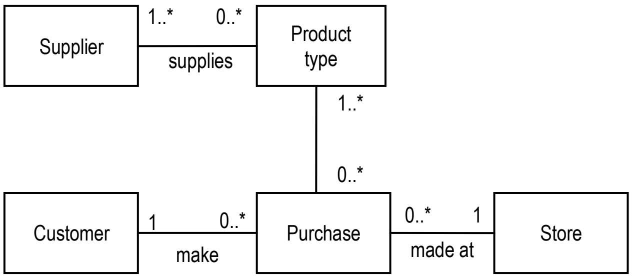

Fig. 2. Normalised conceptual structure

We can simplify the structure as before by replacing the entity types with more general categories. To do this, we observe that customers are people, stores are places, suppliers are organisations, purchases are activities, and product types are categories. The result is illustrated in Figure 3. We now have a more general model with potentially wider applicability. Role-based concepts like *customer* and *supplier* have been replaced by more generic categories and encapsulated in relationships.

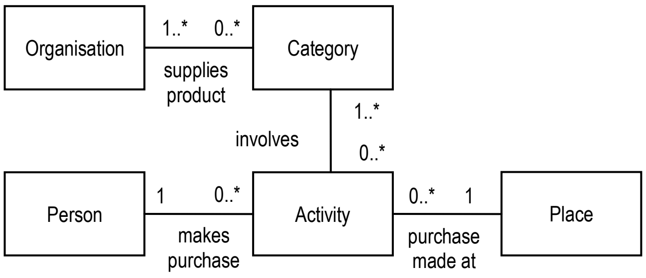

Fig. 3. Roles replaced with archetypal categories

Of course, this model is still subject to volatility, because the relationships are likely to alter over time. If these relationships were implemented in a database structure they would "fossilize" a particular snapshot of the conceptual structure, and make it hard to modify or extend later on. One way of avoiding that is to represent the entity types and relationships as data, using a structure similar to the one shown below.

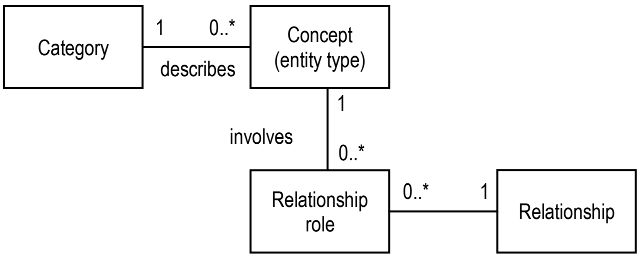

Fig. 4. Concepts and relationships represented as data (for clarity, provision for storage of attributes is omitted)

This provides a structure which is effectively a meta-model; it is designed to store conceptual models as data. We refer to a conceptual model stored in this manner as a *soft schema*. Soft schemas can be stored using any appropriate means, including in databases or as XML. The corresponding data described by each conceptual model can also be stored in a variety of ways, but XML is an obvious candidate, as shown in the example below.

```
<customer category=″person″>
 <name>Joanne Wall</name>
 <id>2012</id>
 <address>43 Tows Str</address>
</customer>
<customer category=″person″>
 <firstname>Maurice</firstname>
 <lastname>Smith</lastname>
 <id>2002</id>
 <address>3 Yannou Street</address>
 <phone>2273034397</phone>
</customer>
```

Fig. 5. Data fragment

Note that this XML fragment exhibits CDI, because it does not conform to any particular conceptual structure. In effect, each instance of data carries its own conceptual structure. The example shows two data instances, and although both refer to the same concept (*customer*), the concept is defined differently in each instance. In a conventional database or application, this would be evidence of a failure of data integrity, and would probably cause the application to fail. But in a system with CDI, it merely reflects the fact that the conceptual structure has evolved over time, or is contingent on context, or data has been merged from heterogeneous applications, or some other circumstance. In other words, such differences in conceptual structure between data instances are natural and entirely to be expected. Any software application which uses this data would be required to cope smoothly with the difference in structure between the two instances.

With appropriate management software (analogous to database management software) domain-level semantic constraints can be enforced including referential integrity, subject to the constraint that data relating to multiple schema versions must be able to co-exist. It is envisaged that this layer of systems software would mediate between the data storage and applications which access it, in much the same way that database management software does.

This ability to store data instances corresponding to multiple schemas alongside one another provides a unique advantage. Because the architecture is not specific to any particular conceptual structure, it allows for the storage of data pertaining to any conceptual structure, and therefore any application domain. The logical consequence is that data could be stored for any *and all* domains using a single datastore. In other words, a single datastore could concurrently hold data corresponding to any number of distinct schemas. By implication, a single application could consult this datastore, responding in real time to the embedded conceptual structures to provide suitable functionality across multiple domains. We refer to applications with this property as *domain independent* [25]. The potential for domain-specific design in such an application would be reduced and the consistency of application design would be increased, relative to current practice. This may or may not be an advantage and is a subject for further research. We note also in passing that the functions of such an application could easily be incorporated into an operating system or other system software.

4 Archetypal Categories as a Basis for Integration

The example above refers to a number of basic-level concepts which are considered relatively invariant. They include *people, places, organisations, activities,* and *categories*. This is not an arbitrary list of concepts; it stems from research into the cognitive aspects of conceptual modelling and system design [26]. The list also includes *documents, physical objects, conceptual objects* and *systems*. These basic-level concepts are termed *archetypal categories*.

According to one view of cognition, meaning is generated in the brain by unconscious feature-driven classification of perceptual inputs on the basis of similarity and associative recall. While it had been thought that the brain's neural networks were structurally indifferent to categories, evidence suggests that the mind has evolved to give preference to certain concepts in particular; examples from different studies include people, activities, tools and locations. It has been proposed that human memory tends to converge on such basic-level concepts, which are neither highly specific nor particularly generic [19]. Physical evidence from brain imaging studies also suggests that we may possess a limited number of hard-wired semantic regions into which perceptions are routed [27], corresponding again to categories pitched at a basic, everyday level [28]. We suggest that IS could exploit the familiarity of these innate categories by storing and presenting data in terms of them. Further, we suggest that their use would make data exchange easier.

Ontologies typically include thousands of classes, but only a subset correspond to basic-level concepts. For example, the ontology SUMO contains the hierarchy *sentient agent* → *cognitive agent* → *human* → *internet user* [29]. The class *internet user* is a role, and the classes *sentient agent* and *cognitive agent* are abstract; this leaves *human* as the only basic-level concept (essentially identical to our *person*). In a similar vein, *animal* might be more easily understood than *organism*, and *man* easier to deal with than *hominid*. In our list of archetypal categories *person* and *organisation* could be replaced by the more general concept *party*, but again this would not be pitched at a basic level and would therefore not be so understandable.

It should now be apparent how CDI and archetypal categories can offer a solution to the problem of conceptual incompatibility, allowing the exchange of conceptually-incompatible data. IS could use a small vocabulary of archetypal categories, reflecting the mind's basic-level concepts. This would provide the common conceptual framework required for meaningful exchange of information [1]. If data is expressed in terms of a small list of shared archetypal categories, it can be understood by both parties in the exchange even if no concepts per se are shared.

The example in Figure 5 is a simple illustration of this principle. Both instances of *customer* refer to the archetypal category *person*. To deal with the data, the receiving application would need to "understand" what people are and how to handle data about people—without expecting any particular conceptual structure attached to instances of the category *person*. The receiving application would thus not need to share a conceptual model or ontology with the sending application. The same argument applies for data corresponding to the other archetypal categories: places, organisations, documents and so on. Implementation of this simple set of archetypal categories in the context of domain-independent applications could therefore offer a convenient "middle road", allowing data to be exchanged meaningfully without the need for complex shared conceptual models or ontologies.

5 Conclusion and Further Work

To summarise, the argument for conceptual data independence is as follows. Interoperability between applications and easy exchange of data are desirable goals, but heterogeneous design makes them difficult to achieve. Standard design practices create ad hoc, incompatible conceptual structures. This was acceptable when there were relatively few applications and change was infrequent. However, as a result of the creation of many applications and increasingly rapid business change, conceptual incompatibility is causing an unacceptable increase in system development, maintenance and integration workloads.

The idea behind CDI is that data is stored and exchanged in a form that is invariant with respect to conceptual structures; each instance of data carries its own conceptual structure, which can be interpreted at runtime. This implies that data corresponding to multiple schemas can co-exist within the same datastore or application. When used in application design, CDI therefore has potential to reduce development and maintenance workloads substantially, because applications do not have to be domain-specific. In effect, one application with CDI could fulfil the function of many of today's domain-specific, non-CDI applications; the result could be a substantial reduction in cost and delay. CDI also has implications for data exchange; any two datasets can be merged without effort if they are stored in a conceptually-independent manner, provided that both use a common set of concepts. The use of archetypal categories provides such a set of common concepts which can easily be implemented in multiple domain-independent applications, because it does not rely on agreement about, and implementation of, complex ontologies.

Research is proceeding into the use of CDI. One project has produced a proof-of-concept software prototype which demonstrates how the need to modify software applications can be avoided as conceptual structures evolve [25]. Work is in progress on usability testing. Next, it is planned to proceed with the development of a fully-featured domain-independent application in order to test the impact of CDI on system maintenance and data integration. Overall, CDI represents a fundamentally different approach to information system construction; further empirical and theoretical research will be needed to explore the significant possibilities that it affords.

References

1. Liebenau, J., Backhouse, J.: Understanding Information: An Introduction. Macmillan, Basingstoke (1990)
2. Sowa, J.F.: The Challenge of Knowledge Soup. Vivo Mind Intelligence, Inc. (2004)
3. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. The International Journal on Very Large Data Bases 10, 334–350 (2001)
4. Hauser, L.: Searle's Chinese box: debunking the Chinese room argument. Minds and Machines 7, 199–226 (1997)
5. Kalfoglou, Y., Hu, B.: Issues with Evaluating and Using Publicly Available Ontologies. In: Chen, C. (ed.) Handbook of Software Engineering and Knowledge Engineering (2006)
6. Taylor, P.: Adhocism in software architecture-perspectives from design theory. In: Proceedings of International Conference on Software Methods and Tools, SMT 2000, pp. 41–50 (2000)
7. Klein, M.: Combining and relating ontologies: an analysis of problems and solutions. In: Workshop on Ontologies and Information Sharing, IJCAI, vol. 1 (2001)
8. Eick, S.G., Graves, T.L., Karr, A.F., Marron, J.S., Mockus, A.: Does code decay? Assessing the evidence from change management data. IEEE Transactions on Software Engineering 27, 1–12 (2001)
9. Fonseca, F.T., Martin, J.E.: Toward an Alternative Notion of Information Systems Ontologies: Information Engineering as a Hermeneutic Enterprise. Journal of the American Society for Information Science and Technology 56, 46–57 (2005)
10. Fensel, D.: Spinning The Semantic Web: Bringing the World Wide Web to Its Full Potential. MIT Press, Cambridge (2005)
11. Miller, R.C.B.: railway. com. Institute of Economic Affairs, London (2005)
12. Bartky, I.R.: Selling the True Time: Nineteenth-century Timekeeping in America. Stanford University Press, Stanford (2000)
13. Siau, K.L., Chan, H.C., Wei, K.K.: Effects of query complexity and learning on novice user query performance with conceptual and logical database interfaces. IEEE Transactions on Systems, Man and Cybernetics, Part A 34, 276–281 (2004)
14. Maathuis, I., Smit, W.A.: The battle between standards: TCP/IP Vs OSI victory through path dependency or by quality? In: The 3rd Conference on Standardization and Innovation in Information Technology, pp. 161–176 (2003)
15. David, P.A.: Clio and the Economics of QWERTY. The American Economic Review 75, 332–337 (1985)
16. Craighead, C.W., Patterson, J.W., Roth, P.L., Segars, A.H.: Enabling the benefits of Supply Chain Management Systems: an empirical study of Electronic Data Interchange (EDI) in manufacturing. International Journal of Production Research 44, 135–157 (2006)
17. Hendler, J., Berners-Lee, T., Miller, E.: Integrating Applications on the Semantic Web. Journal of the Institute of Electrical Engineers of Japan 122, 676–680 (2002)
18. Khare, R., Çelik, T.: Microformats: a pragmatic path to the semantic web. In: Proceedings of the 15th International Conference on the World Wide Web, pp. 865–866. ACM, Edinburgh, Scotland (2006)
19. Pansky, A., Koriat, A.: The Basic-Level Convergence Effect in Memory Distortions. Psychological Science 15, 52–59 (2004)
20. Heath, T., Motta, E.: Ease of interaction plus ease of integration: Combining Web 2.0 and the Semantic Web in a reviewing site. Web Semantics: Science, Services and Agents on the World Wide Web 6, 76–83 (2008)

21. Graham, I., Spinardi, G., Williams, R., Ivebster, J.: The dynamics of EDI standards development. Technology Analysis & Strategic Management 7, 3–20 (1995)
22. Musen, M.A., Lewis, S., Smith, B.: Wrestling with SUMO and Bio-ontologies. Nature Biotechnology 24, 21 (2006)
23. Guzman-Arenas, A., Olivares-Ceja, J.M.: Measuring the understanding between two agents through concept similarity. Expert Systems With Applications 30, 577–591 (2006)
24. Wieringa, R., de Jonge, W., Spruit, P.: Roles and dynamic subclasses: a modal logic approach. In: Proceedings of European Conference on Object-Oriented Programming (1994)
25. Kapros, E.: Multi-component Evaluation of an Adaptive User-interface for a Generic Application. In: Workshop on Experience, Usability, and Functionality, Irish HCI Conference 2009, September 17-18 (2009)
26. McGinnes, S., Amos, J.: Accelerated Business Concept Modeling: Combining User Interface Design with Object Modeling. In: Harmelen, M.V., Wilson, S. (eds.) Object Modeling and User Interface Design: Designing Interactive Systems, pp. 3–36. Addison-Wesley, Reading (2001)
27. Mason, M.F., Banfield, J.F., Macrae, C.N.: Thinking About Actions: The Neural Substrates of Person Knowledge. Cerebral Cortex 14, 209–214 (2004)
28. Eysenck, M.W., Keane, M.: Cognitive Psychology: A Student's Handbook. Psychology Press, UK (2005)
29. Niles, I.: Mapping WordNet to the SUMO Ontology. In: Proceedings of the IEEE International Knowledge Engineering Conference, pp. 23–26 (2003)