Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 94 (2004) 51–58
www.elsevier.com/locate/entcs

Data Store Models are Different Than Data Interchange Models

Michael Blaha[1]

OMT Associates Inc, Chesterfield, Missouri, USA

Abstract

The model of a data store is much different than the model of a data interchange. The model of a data store is driven by the needs of building an application. In contrast, the model of a data interchange is driven by the needs of conveying data among applications. A past project for integrating chemical engineering data illustrates the ill consequences of overlooking the differences.

Keywords: data interchange, data store, metamodel, model, PDXI, repository

1 Data Store vs. Data Interchange

A *data store* is a place where persistent data is managed for an application. A data store lies at the heart of an application, defining its critical concepts, storing its data, constraining its data, and facilitating its behavior. A data store must be carefully managed to ensure correctness and provide an application memory.

In contrast, a *data interchange* provides the means to convey data among applications. A data interchange is transient and exists solely to move data from a source to targets. The source application must translate the data it is sending into the data interchange format. Similarly, a target application must use the data interchange format to populate its own internal data store. Thus a data interchange mediates applications and stands apart from them.

[1] Email: blaha@computer.org

doi:10.1016/j.entcs.2004.04.001

	Data store model	Data interchange model
Purpose	Defines basic concepts and relationships for building an application	Moves data among applications; must be translated to and from the application format
Scope	Particular to one application	Spans multiple applications
Data lifetime	Long (persistent data)	Short (transient data)
Data size	Small to large	Small to medium
Read access	Random; queries can be complex	Sequential
Update access	Random	No update, other than creating interchange file
Concurrency	Multiple users	Single user

Table 1
Requirements. A data store has much different requirements than a data interchange model

	Data store model	Data interchange model
Schema size	Large	Small
Schema complexity	Medium; reflects application concepts	High; involves metadata and data
Flexibility	Dedicated to one application	Must accommodate future applications
Storage medium	Usually a database; sometimes files	Usually a file
Key concerns	Enforcement of data quality despite updates; high performance	Format must be easy to parse

Table 2
Resulting architecture. The requirements lead to different architectural decisions

The requirements for managing data internal to an application are much different than the requirements for moving data among applications as Table 1 shows. Table 2 shows the consequences of the requirements on architecture decisions.

An interchange model does not imply a specification for building an application. Rather, a well-designed interchange model would normally be unsuitable for building an application. An interchange model must be small in structure and straightforward to parse. It need not be concerned with enforcing data quality.

Given the need for small schema size and flexibility, a typical interchange model combines metadata and data. This can make them tricky to define and understand.

2 An Example – the PDXI project

We will clarify our ideas by discussing a past project. The Process Data eXchange Institute (PDXI) was formed in the early 1990s by twenty companies, mostly petrochemical companies, under the auspices of the AIChE (American Institute of Chemical Engineers) [2]. The purpose of PDXI was to develop a means for exchanging data among chemical engineering design applications.

The project contract was awarded in competitive bidding to a team led by Neil Book of the University of Missouri, Rolla. Members of the team came from various affiliations and were Michael Blaha, Jim Fielding, Barbara Goldstein, John Hedrick, Rudy Motard, and Ollie Sitton.

Initially there was a concern about awarding the contract to a university-led team. The industrial sponsors were concerned that a university-led team would not be business like and complete the project. However, those fears became unwarranted as the contract team gelled and worked well together.

The contract team prepared about 150 pages of object-oriented class models, covering chemical process design – topics such as chemical properties of substances, equipment design, and chemical process simulation. These models were intended as the specification of a data exchange standard. Unfortunately, the industrial sponsors were not prepared to receive and act on all these models. Ultimately, the PDXI project led to limited progress.

In retrospect, one problem with our deliverables is now clear. The 150 pages of models that we delivered was much too large for an interchange standard. At thetime no one realized, but we had made the mistake of confusing an application model with an interchange model.

Figure 1 shows a small excerpt of a data store model for equipment. (The PDXI equipment model was more than 50 pages long.) There is a wide and deep taxonomy and the superclass-subclass structure organizes the many pieces of data.

Figure 2 shows a data interchange model for equipment. An object has a class that determines attributes for which values can be stored. The classes are organized into a hierarchy using single inheritance. *Equipment* inherits from *Object*. This model is also abridged and we could elaborate it into a more complete equipment metamodel involving aggregation, association, and other aspects.

Figure 1 has ten classes, each of which is a record in *Class* for Figure 2. Figure 1 has three generalizations that would become three records in *Generalization* for Figure 2. A single *CentrifugalPump* object in Figure 1 would lead to ten *Value* records for Figure 2.

You can see that the data store model is tangible and directly corresponds

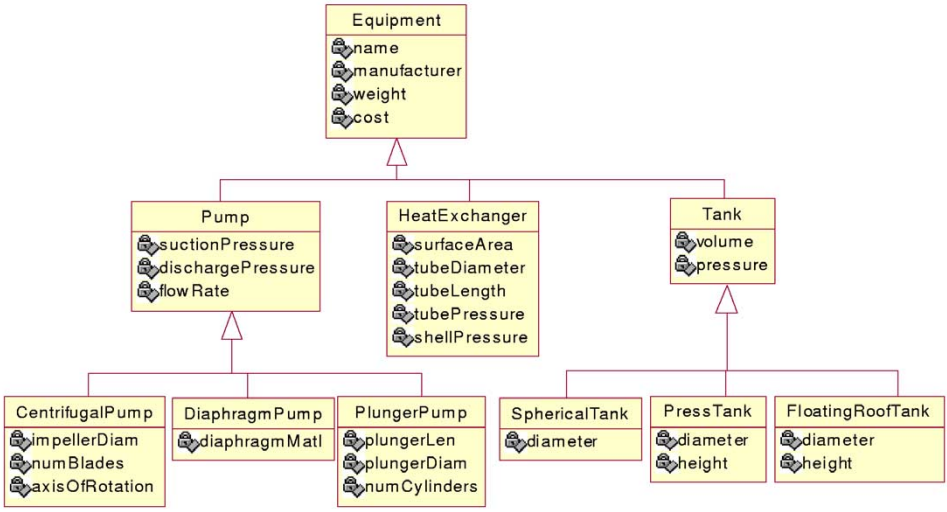

Fig. 1. Example: Data store model for equipment

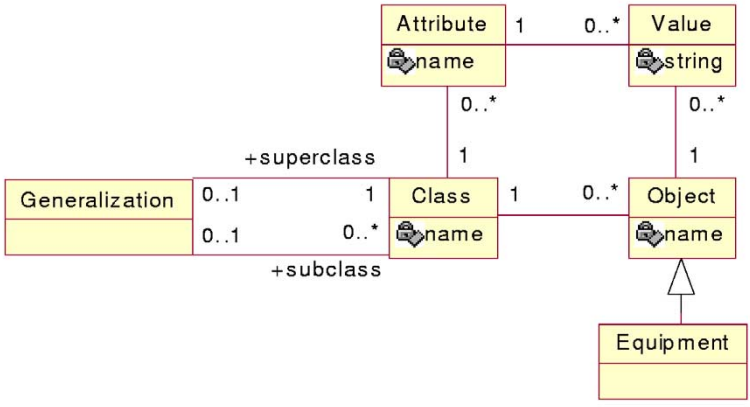

Fig. 2. Example: Data interchange model for equipment

to the application. In contrast, the data interchange model is abstract, combining data and metadata. The data store model can be quite large, but is easier to understand. The data interchange model is small, but can involve numerous records with its population.

A data interchange model should involve metadata and be abstract, so that it is more flexible for change. For example, the addition of a *DistillationColumn* class as a sibling to *Pump*, *HeatExchanger*, and *Tank* in Figure 1 would necessitate the addition of a class with its various read and write methods. In contrast, the *DistillationColumn* class would cause no change to the

structure of Figure 2 and would only add some more metadata. So Figure 1 requires structural change and programming for new kinds of equipment while Figure 2 is unlikely to require any changes.

In retrospect Figure 2 would have been the right way to model equipment for the PDXI project. But we did not realize it at the time and used the Figure 1 approach.

3 Lessons Learned from the PDXI Project

The project had a number of successes, but there were things we could have done better.

Distinction between application and interchange models. Although we did not recognize the distinction at the time, something was amiss with our approach. 150 pages of models is too large for an interchange model. It is only in retrospect that we clearly recognize our mistake as this paper explains.

Confusion about metamodels. The PDXI sponsors had difficulty understanding metamodels and this compounded our tendency to make the models too large. One portion of the PDXI models was for equipment – the industrial sponsors liked these models even though they were too large and unwieldy for an interchange standard.

Another portion of the PDXI models covered physical properties (melting point, vapor pressure, specific gravity, smoke point, and many others). We happened to build a metamodel for this portion and the corporate sponsors did not like it. They kept pressing for something more tangible. In retrospect, the industrial sponsors were uninformed. They wanted a data interchange model for their business purposes but had no idea what this should look like.

Uneven approach. As previously mentioned, we prepared a tangible model for equipment and a metamodel for physical properties. We did not realize it at the time, but such a variation in abstraction is, in itself, a troublesome sign. The different levels of abstraction were a side effect of one group of persons working on equipment and another group working on physical properties. We should have reconciled the two approaches and chosen a uniform abstraction level. Our lack of realization was compounded by the size of the effort and the inability of our industrial sponsors to receive highly abstract models.

Ownership. We had a problem with handing off ownership. The industrial sponsors were willing to pay for the interchange models, but did not want to

support them. Some vendors were willing to support the models, but that was undesirable, because the interchange models were intended to bridge vendors and be apart from their individual interests. Ultimately, a dominant vendor in chemical engineering software received the models and supported them – that was the best compromise that we could devise.

Orphan technology. Nearly all of the PDXI funding came from petrochemical companies. Their primary interest is in petrochemical technology. The data interchange problem is important to them, but still is not a core interest. It is difficult to get corporate support for something that is not a core competency. As a consequence, our supporters were only from the first and second levels of management. They had little funding authority, limiting the PDXI budget. In contrast, a few years later the European Community initiated a similar project with heavy government funding, much more than that of PDXI.

Coordination. With any kind of standard there are a host of conflicting interests. We had difficulty coordinating everyone – primarily the modeling team and the many sponsors – and reconciling various points of view. We never did receive full sign-off (or rejection) of many of our models.

Neither right nor wrong. The models in Figure 1 and Figure 2 are both correct. So the issue in this paper is not a matter of "correct" or "incorrect" models. Rather the issue is a matter of scope and a models purpose. From additional experiences with other projects, we have learned that focusing on the scope and purpose is essential to constructing useful models.

Documenting requirements. In a later, unrelated project, we learned how to present metamodels better. We built a metamodel as the project requirements demanded and then built application models to populate the metamodel. We showed the application models to end users so that they could verify their content. We told the users about the metamodels but did not show them in depth – we portrayed metamodels as a computing mechanism for realizing the application models. In a sense, the use of a metamodel and application models is a multi-layered data model, as [3] explains.

4 Repository Technology

The literature has a number of papers on repository technology which is a related topic to data interchange. A repository is a database that holds information for software engineering tools. A repository serves as an intermediary for moving data among the various tools and provides storage for general-purpose data apart from the needs of particular tools. Software engineering

	Interchange model	Repository
Intersection data	Both emphasize the intersection data of applications (their common data)	
Metadata	The underlying models involve much metadata.	
Heterogeneity	Both must integrate a wide variety of applications, some of which may perform the same function.	

Table 3
Similarities — repository vs. interchange model

	Interchange model	Repository
Persistence	Short lived	Long lived
Queries	No queries; merely transits data	Subject to sophisticated queries
Access	Single user access	Multiple concurrent users
Data management	Files	Database (because long-lived and multi-user)
Notification		May notify applications of changes to data for which they are registered
Versions	A snapshot in time and need not handle versions	Versions of metadata and changes over time

Table 4
Differences — repository vs. interchange model

tools import and export data from the repository. Reference [1] discusses requirements for repository software.

As you can see from the definition, there is overlap between the purpose and function of a repository and that of an interchange model. Table 3 notes the similarities. Table 4 notes the differences.

5 Conclusion

Based on our experience, we have the following advice.

Size. Interchange models should be kept small (ideally no more than several pages long).

Abstraction. Interchange models should be highly abstract. It is a separate problem to get people to understand them.

Flexibility. A small, abstract model is most likely to serve the interchange needs of future unforeseen applications.

True integration. Separate modeling efforts must be deeply reconciled. It is not satisfactory to just have the union of everything.

We see software data exchange – exchange between compilers, debuggers, visualization software, reverse engineering tools, and so forth – as completely analogous to the chemical engineering example. There are many applications to integrate with various amounts of overlap. It is better to use a small model that abstracts across the anticipated differences than to write tedious specific code for every need.

References

[1] Michael Blaha, David LaPlant, and Erica Marvak. Requirements for repository software. *Fifth IEEE Working Conference on Reverse Engineering*. Honolulu, Hawaii, October 1998, 164-173.

[2] Neil Book, Oliver Sitton, Rodolphe Motard, Michael Blaha, Barbara Maia-Goldstein, John Hedrick, and James Fielding. The road to a common byte. Chemical Engineering magazine. September 1994.

[3] Anthony Cox and Charles Clarke. Multi-layered data modeling. *Workshop on Meta-Models and Schemas for Reverse Engineering, Victoria, British Columbia, November 2003. Electronic Notes in Theoretical Computer Science* (this volume), 2004.