# An Architecture for Universal CAD Data Exchange

Ari Rappoport*

## ABSTRACT

Parametric feature-based CAD data exchange is one of the most important open problems in solid modeling. The problem is significant and challenging both scientifically and commercially. In this paper we present a very general outline of the Universal Product Representation (UPR) architecture, which provides universal support for all data levels employed by today's CAD systems. The architecture has been implemented with successful results.

## Categories and Subject Descriptors

D.2.11 [**software architectures**]: domain specific architectures; D.2.12 [**interoperability**]: data mapping; I.3.5 [**computational geometry and object modeling**]: Breps, CSG, solid, and object representations, geometric languages and systems; I.3.6 [**methodology and techniques**]: graphics data structures and data types, languages, standards; J.2 [**physical sciences and engineering**]: engineering; J.6 [**computer-aided engineering**]: CAD, CAM.

## General Terms

Design, Standardization, Languages, Verification.

## Keywords

CAD Data Exchange, Parametric Data Exchange, Feature-Based Data Exchange, Universal Data Exchange.

## 1. INTRODUCTION

In this paper we address CAD Data Exchange (DE), which is one of the most important open problems in solid modeling. Data exchange is an extremely significant issue, both scientifically and commercially. Scientifically, parametric feature-based DE is challenging in terms of semantics, algorithms and data structures of geometric operations and algorithms for mapping between them. Commercially, DE is currently the preferable approach to CAD interoperability, a major topic in engineering collaboration. An in-depth discussion of the problem motivation is given in [Rappoport03].

In this paper we present a novel architecture for CAD data exchange. The architecture is designed to support all of the important data levels employed by current CAD systems, including the parametric feature-based design level. It is explicitly designed to handle the technical and business considerations that affect data exchange today.

*School of Computer Science and Engineering, The Hebrew University, Jerusalem Israel, and Proficiency Ltd. arir@cs.huji.ac.il.


*SM'03,* June 16–20, 2003, Seattle, Washington, USA.


## 2. PREVIOUS WORK

The main data exchange solutions in use today are IGES and STEP [Bloor95]. Both are data standards created by committees. IGES originally supported drawings and wireframes, and was later extended to support surfaces and solids. STEP was designed with the intent of being flexible with respect to future enhancements. It includes its own data modeling language and a modular architecture. It is mostly used today for exchange of 3-D boundary representations and assemblies, and some of its industry specific extensions are also in use. Neither IGES nor STEP support the parametric feature based design modeling paradigm. There is a STEP specification for the representation of manufacturing features, which is not directly relevant to data exchange. There is work in progress for supporting parametric design in STEP [Pratt01], but it is a long way until it is put to use.

At this time the main detailed published proposal for feature based DE is the Erep file format [Hoffmann93a]. The Erep project was a pioneering research project whose main direction was well-defined specification of feature based design, and the Erep file format can be viewed as a DE solution. The project's contribution to theory was ground breaking, even though it has dealt with only a few of the CAD design features. However, it still had not had any practical effect on data exchange.

Engen was a STEP-related project for addressing feature based DE [Shih97]. A prototype for limited DE of parametric 2-D sketches was reported, but no progress had been reported beyond that.

Although fundamentally different in many aspects, IGES, STEP and Erep are very similar when viewed as data exchange *architectures*. All of them rely on a documented, common data model. In all of them, the architecture centers around a file format that is the storage embodiment of the data model. There are no built-in mechanisms to

deal with incompatibilities or failures. All of them thus operate in an ideal world that assumes that all parties that interface with them operate perfectly and according to an agreed specification. The analysis presented in the next section shows how problematic this mode of operation is for data exchange, both practically and theoretically.

# 3. GUIDING PRINCIPLES

Our DE architecture is based on recognizing the following major observation and respecting its implications: *there are and will always be both functional and implementational incompatibilities between CAD systems.* A major goal in the design of our architecture is to find ways to overcome this problem. In this section we discuss the reasons for those various incompatibilities and their nature.

## 3.1 Inherent Functional Incompatibilities

A universal CAD data exchange architecture must deal with the fact that CAD system functionalities are not fully compatible. The domain of engineering software is extremely rich; it is always possible to think of a new function that can be put to use for assisting some engineering activity.

Such new functions are continuously being introduced by CAD systems, because market dynamics encourage CAD vendors to offer capabilities that are not offered by their competitors. Competition and the domain complexity ensure that such inherent functional incompatibilities will be present in the forseeable future. It is not realistic to expect that CAD vendors will trim down the functionality of their products only in order to fit a particular DE solution.

## 3.2 Semantics Known at Runtime Only

As convincingly noted in [Hoffmann93b], CAD system functionality is not defined rigorously anywhere. Parametric design features are documented in a user's guide, and their geometric semantics are nowhere specified formally.

This problem is not due to incompetence or lack of cooperation on the side of the CAD vendors. Formal semantics of geometric operations is a challenging theoretical issue whose solution is expected from academic research. CAD vendors are thus forced to ultimately rely on actual coding for determining the full semantics of geometric operations.

In many cases the general semantics of a feature can be quite easily defined formally, but not in a way that covers all possible input configurations. For example, 'constant radius fillet' (or round) is theoretically well understood in terms of 3-D pointsets. However, in practice, the exact nature of the surfaces and topology created by a particular constant radius round feature (and whether or not it will work successfully) strongly depend on the relationship between the solid's local and global geometry and the size of the radius.

As a result, the precise geometric semantics of feature operations can only be inferred from the actual behavior of the CAD system.

## 3.3 Implementational Incompatibilities

Computer science has still not found a practical and theoretically fullproof method for modeling geometric entities. Real numbers are modeled using floating point arithmetic, which means that even the most fundamental computations can produce inconsistent results (e.g., the 'is point on line' query regarding the intersection point between two lines.)

CAD systems are implemented by different software teams. Even if two CAD systems allegedly support a data type with the exact same semantics, they will utilize differing implementations. As a result, exchange of any geometric data item between the two systems may fail due to tolerancing problems. This issue must be taken into account by any CAD DE architecture.

## 3.4 Bugs

Software systems always contain bugs. A data exchange architecture provides integration between software systems, and should be as resistant to their bugs as possible.

## 3.5 Economic Feasibility of Implementation

The implementational effort of our architecture must be linear (or almost linear) in the number of CAD system versions supported. Otherwise it will not be feasible to sustain support for a solution over time.

## 3.6 Implications

Inherent functional incompatibilities prevent usage of a common data model. The fact that some functional semantics can be discovered only during runtime, and the existence of implementational differences and bugs, mean that we cannot specify a single DE flow in advance. The architecture must be capable of recognizing runtime failures and recuperating from them.

# 4. THE UPR ARCHITECTURE

In this section we provide a very general description of a novel architecture for complete CAD data exchange, called the *Universal Product Representation (UPR).*

## 4.1 A Star Architecture

To ensure economic feasibility, we, like previous DE architectures, use a star architecture. Each CAD system interfaces with a central data repository through export and import modules (Figure 1). Unlike previous approaches, this does not mean that we enforce an identical data model on all participating systems (see below).

## 4.2 Universal Support of Data Types

To address inherent functional incompatibilities, we take the exact opposite route to the 'least common denominator' approach of previous solutions: the UPR data structures are flexibly designed to support *all possible data types*; they can represent the *union* of the data types supported by CAD systems, not only their intersection.

## 4.3 Data Unification

To optimize the UPR data structures, during design we study the semantics of each data type of the supported CAD systems and identify data types that are of similar semantics. Initial semantics assumptions are made according to the CAD system's user guide, and are modified after extensive empirical usage of the CAD system by experienced users. Data types that are finally considered similar share data structures.

Data unification is only an optimization. When data types are incompatible, they are not unified and are handled through rewrites (see below). This stands in sharp contrast to previous methods, in which each and every data type is 'unified' by adopting a 'standard' definition.

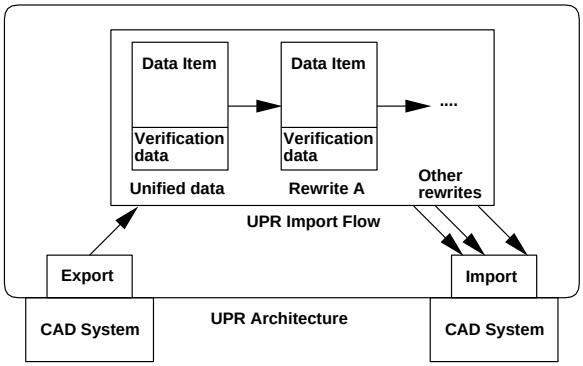


**Figure 1: For each data item, the UPR architecture stores import flow rewrites and verification data.**

As a simple example, take the very common case of CAD systems using different terms to denote semantically identical operations (e.g., 2-D Sketch and 2-D Section, Round and Fillet). In this case unification simply amounts to selecting one of these terms and storing the data under its title. Another simple example is given by a 'triangle' object represented by three vertices: one system may use a clockwise orientation while another may use a counter-clockwise one. The UPR would arbitrarily select one of these options; import procedures would employ the appropriate orientation.

In practice, data unification requires unifying data structures that are not formally mathematically equivalent. For example, as mentioned above, almost all of the design features provided by CAD systems are not formally specified. While their 'ordinary' semantics may possibly be understood and expressed formally, their behavior under various configurations is in many cases hard to define. Nonetheless, we may allow their unification provided that their 'inordinary' behavior occurs in a small number of cases, and relying on the rewrite mechanism (see below) to handle these cases.

Data unification serves to reduce the number of disparate data types in the UPR. The exact magnitude of this reduction depends on the functional similarities between the CAD systems and on the degree of practical compromising exhibited by the UPR designer. Since almost all modern CAD systems are based on the same fundamental paradigm and generally provide similar functionality to their users, data unification can be expected to produce effective results.

## 4.4 Rewrites and Import Flow

As emphasized above, a major guideline of the UPR is addressing functional and implementational incompatibilities. We answer this design goal by the concept of a *rewrite* of a data item as another data item. A data item rewrite is used during import, in two cases: (1) when the target CAD system does not have a compatible data type, and (2) when import has failed for some reason. The implied import flow is: if you do not succeed (for any reason) in importing the unified data item into a CAD system, rewrite it and try again.

A set of data type rewrites is attached to any unified type (Figure 1). Rewrites usually modify the data itself, but they may also be purely functional, representing a different algorithm for importing the same data. When a rewrite modifies the data, the data type conversion is quite possibly, but not always, semantically lossy.

There are many situations where data type rewrites are highly applicable in the CAD domain. The main example involves loss of associativity in some form. As an example take the common 'Extrusion' parametric feature. The Extrusion operation takes a 2-D set and extrudes it in 3-D space according to a given axis and a given size. The Extrusion feature usually offers many different parameters. Two useful parameters are related to the extrusion size: the size can be given explicitly as a distance (this is sometimes called a 'blind' extrusion), or it can be given by specifying the extrusion to end when the extruded 2-D set meets a certain geometric entity already present in the model (say, a Brep face of the solid model). This latter usage is termed 'associative' because the extrusion size is automatically modified when the geometric location of the target entity is modified. Suppose now that the unified data type representing the Extrusion operation contains a 'target entity' parameter. Suppose further that this capability is lacking in some CAD system. A data rewrite of the unified Extrusion can be defined by replacing the 'target entity' parameter by an 'absolute size' parameter. In effect, an 'Associative Extrusion' data type is converted into a 'Blind Extrusion' data type. Such a conversion makes sense from a data exchange perspective when the two operations are expected to possess similar geometric semantics (that is, when the geometric changes performed on the model are identical).

In many cases, there exists a natural *hierarchy* of data rewrites. In the Extrusion example above, the rewrite into a blind extrusion is not the last one that can be attempted. If a blind extrusion does not succeed, it can be rewritten into a 'glue faces' operation. This operation can be implemented by computing the set of faces generated by the original feature (in this case, if the associative extrusion and the blind extrusion generate an equivalent set of faces) and gluing them into the model. This rewrite is possible when the target CAD system supports a 'glue faces' operation.

Note that in the two given examples, an actual execution of the rewrite requires data that is not necessarily exported into the unified data type. In the first example, we would need the length of the extrusion, and in the second example the set of faces generated by the extrusion. There are three main alternatives for dealing with such data: it can be computed during export, requested during run-time from the CAD system from which the data had been exported, or computed during import. The last alternative is usually not feasible, because the data usually depends on internal implementation in the source CAD system. The first alternative is the most general but also the most demanding in terms of export efficiency and storage size. The second alternative is the most flexible but it requires access to the source CAD system during import into the target CAD system, which complicates the operating environment of the architecture as well as possibly increasing its cost due to the additional CAD licenses required.

The rewrite hierarchy is not fixed. It may be the case that a user would prefer a particular rewrite for one project and a different one for another project. In this case the sequence of rewrites that are attempted at runtime can be determined by configuration data that can be specified by users prior to the execution of the data exchange operation.

Note that rewrites enable a direct interface between pairs

of CAD systems when this is of advantage, even though the architecture itself is a star one. In order to provide such a direct interface, all we need is to add a 'source target CAD pair' field to the rewrite. When the import flow mechanism is faced with the need to select a rewrite, it can try to first identify the presence of a relevant field of this sort, and invoke the corresponding rewrite before the invocation of other rewrites. Such a capability might be useful for optimization purposes or for unique requirements of a particular project.

## 4.5 Dynamic Import Verification

In order to be able to implement the outlined import flow, which involves iterative attempts of data rewrites and renewed imports, it is essential to be able to recognize import failures. Each data type stored in the UPR, including the unified data types and all of their rewrites, need to have success verification mechanisms associated with them. In some cases, success verification is only a matter of obtaining a binary answer from the CAD system (yes or no). In most cases, however, success verification requires additional data to be stored within the data type. For example, the success of an Extrusion feature can really only be verified by comparing the geometry generated by it in the source CAD system with that generated by the target CAD system.

Data import verification in general, and feature verification in particular, are topics that merit a detailed discussion which is certainly beyond the scope of this paper. For the purposes of this paper it is sufficient to recognize that provisions for storing verification data must be made within the UPR data structures, and that import verification is a basic ingredient of the import flow execution.

In practice, it is sometimes possible to relax those strict verification requirements. For example, suppose that we have no rewrites for design features and we must invoke a 'final geometry' global solid rewrite whenever there exists a single feature that does not produce accurate geometry. In this case we do not need verification data for every feature, and it is sufficient to store verification data only for the final, global geometry.

Note that the actual cause of failures is of implementational importance, because dealing with CAD system bugs requires implementation mechanisms that are different from those required by functional incompatibilities.

## 4.6 Summary

In order to effectively communicate with the reader, in this section we have presented the main concepts behind our architecture in a gradual manner. Following is a brief summary of the resulting architecture.

Our data exchange solution uses a star architecture with a central UPR kernel that contains data structures for representing the union of the data and operation types supported by CAD systems. In order to take part in the architecture, a CAD system needs to implement two modules: export and import. The export module moves data items from the CAD system into the appropriate data types, performing data unification when needed (or as a post-process). The import module examines the set of data items that need to be exchanged. For each data item, it locates an import procedure corresponding to the item's data type, and invokes it. Success of import is verified immediately afterwards; if it has failed, another import procedure is located according to rewrites present in the UPR and configuration data. That import procedure is invoked, repeating the process until success is reported. The key concepts here are universal data type support, rewrites, and dynamic import verification.

For the sake of concise communication, we refer to the architecture as 'the UPR architecture'. The term UPR directly communicates the idea of representing the union of data types that are present in the product engineering world, but it does not directly convey the other main ideas of the architecture. However, we have selected it as our key term because it is short and at least partially descriptive, recognizing that it is not feasible to attempt to directly convey too many concepts by a single term.

# 5. RESULTS AND DISCUSSION

The UPR architecture has been implemented at Proficiency Ltd. The current implementation supports the five high-end CAD systems in the market: Catia 4, Unigraphics, I-DEAS, ProEngineer, and Catia 5. Several versions and a majority of the design features of each system are currently supported. The runtime architecture is a distributed architecture having three layers: users that access the system through a web based user interface; a web server that distributes jobs between agents; and computational export and import agents that interface with the CAD systems.

The number of DE operations executed through the system so far is well over 100,000. The DE quality that has been achieved constitutes a dramatic improvement over previous data exchange solutions. Currently, the data exchange quality is very close to a 100%, of which 70%-100% is fully parametric feature-based, depending on project type (different projects use different modeling techniques and features). Thus, we have demonstrated the first working solution for feature based data exchange, and it is of proven practical value to real engineering projects.

Space constraints on this paper had forced us to give here only a very general description of the architecture. There are many interesting conceptual, algorithmic and implementational details that have not been discussed. We intend to report upon these in future papers.

**Acknowledgement.** The Proficiency UPR implementation is headed by Alex Tsechansky. The content of this paper is patent pending.

# References

[Bloor95] Bloor, M.S., Owen, J., Product Data Exchange, UCL Press, University College London, Gower Street, London, 1995.

[Hoffmann93a] Hoffmann, C.M., Juan, R., Erep, an editable, high-level representation for geometric design and analysis. In: P. Wilson, M. Wozny, and M. Pratt, (Eds), *Geometric and Product Modeling,* pp. 129-164, North Holland, 1993.

[Hoffmann93b] Hoffmann, C.M., On the semantics of generative geometry representations. *19th ASME Design Conference,* Albuquerque, New Mexico, September 1993.

[Pratt01] Pratt, M.J., Anderson, B.D., A shape modelling applications programming interface for the STEP standard. *Computer-Aided Design* 33:531–543, 2001.

[Rappoport03] Rappoport, A., The Universal Product Representation (UPR) Architecture for CAD Data Exchange. Technical Report, 2003.

[Shih97] Shih, C.-H., Anderson, B., A design/constraint model to capture design intent. *Solid Modeling '97*, ACM Press, pp. 255–264, 1997.