# A Guide to Property-Based Interoperability

Morad Behandish[1,2] and Vadim Shapiro[1,3]

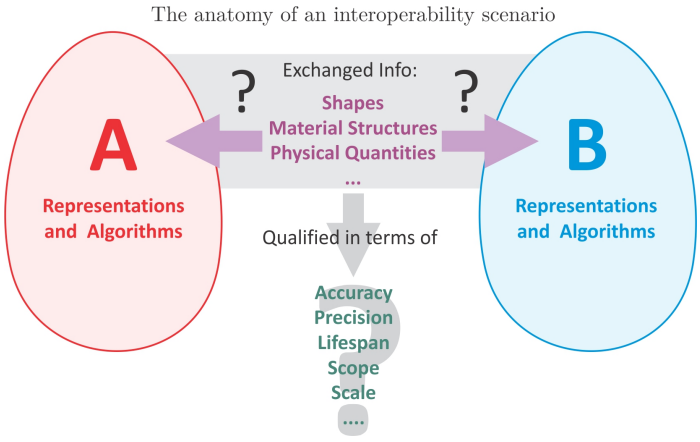
[1]International Computer Science Institute, [2]Palo Alto Research Center, [3]University of Wisconsin–Madison

July 23, 2017

Limited or lacking interoperability has emerged as a central unsolved technical problem in computational design, synthesis, analysis, optimization, and manufacturing. In addition to being a significant technological barrier, it has become a major economic problem within the past two decades, costing the US manufacturing industry billions of dollars every year.

This guide aims to introduce the reader with challenges and opportunities that are common to most interoperability scenarios in computational design and manufacturing. Its main purpose is to provide the engineers, researchers, and other technical professionals with a systematic framework to uniformly think about, formulate, and solve interoperability problems, validate the solutions, and conceptualize extensions. The central tenet of the framework is the notion of correspondence between representations and algorithms that are deemed interchangeable with respect to explicitly declared properties. The concept of 'property-based' interoperability is introduced in terms of the invariance of essential properties in a given data exchange scenario. The main focus will be on geometric properties, which serve as a surrogate to various physical, material, fabrication process, and other properties pertinent to design and manufacturing applications. We provide concrete use-case scenarios for geometric interoperability to serve as examples of how the generic framework is instantiated into different applications and solution methods.

Sections marked by ★ provide more in-depth discussions that can be skipped on a first reading.



# Contents

# 1 An Interoperability 'Dilemma'

Modern design and manufacturing suites have evolved into massive information systems that are computationally intensive, heterogeneous (in scope, domain, and functionality), distributed (in space and time), multi-scale, multi-physical, and interactive. In response to the increasing complexity of the individual components ranging from design tools, simulation codes, and topology optimizers to manufacturing and assembly planners, the subtasks have progressively grown more specialized within their different domains of expertise. Consequently, a bewildering variety of models and representations continue to emerge, in order to support information processing about shape, motion, material structures, physical behavior, design intent, manufacturability criteria, assembly constraints, tolerance specifications, process parameters, and other engineering information whose formal properties are not fully understood.

The lack of proper formalism dragged alongside a continuous upsurge of innovation and specialization leads to nontrivial 'interoperability' problems across disparate computational implementations. Interoperability is a requirement for systematic, modular composition. Problems as diverse as data exchange, substitution, integration, reconfiguration, and reuse of software components in a component-based [23] and service-oriented [19] architectures for computational design and manufacturing are all contingent upon a formal understanding of interoperability.

Moving forward, we find ourselves on the horns of a major dilemma: we can either try to restrict the types of information constructs (i.e., representations and algorithms) by premature standardization, facilitating interoperability while stifling innovation; or we can continue to innovate without restraint, and deal with the ever more challenging problems of interoperability.

## 1.1 What is Interoperability?

A common informal definition of 'interoperability' refers to the ability of a system, whose interfaces are completely understood, to communicate and work with other products or systems, present or future, without any restricted access or implementation [1].

In the context of computational design and manufacturing, interoperability subsumes the aforementioned problems of data sharing, exchange, and translation, as well as the problems of systems integration. Limited or lacking interoperability has emerged as a central unsolved technical problem in research, development, maintenance, scalability, and security of such systems, with crippling effects on further advances in conceptual design, synthesis, analysis, optimization, manufacturing planning, and productivity gains. In addition to being a significant technological barrier, it has become a major economic problem within the past two decades, costing the US manufacturing industry billions of dollars every year [11].

From an operational perspective, one can speak of the following classifications:

- 'Syntactic' versus 'Semantic' Interoperability:
  - Syntactic interoperability is limited to sharing and exchanging information using a compatible language, when consistency of semantics is either trivial or taken for granted.
  - Semantic interoperability is concerned with consistent interpretation of the exchanged information, with or without identical syntax, and subsumes syntactic compatibility as a special case (i.e., simplistic solution).[1]

[1] According to the IEEE Standard Computer Dictionary [1], the former is also referred to as 'compatibility', while the term 'interoperability' alone often implicitly implies semantic interoperability, and shall be taken as such, unless otherwise specified explicitly, hereafter in this guide.

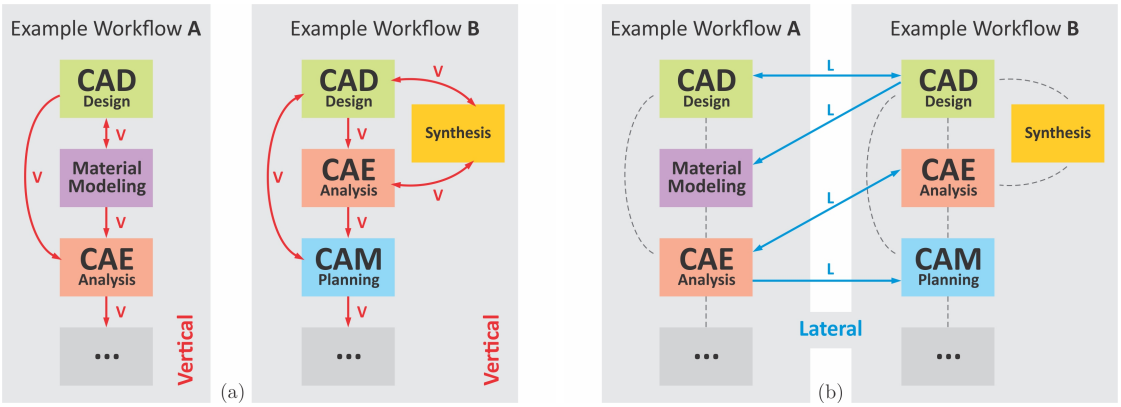



Figure 1: Interoperability can refer to (a) 'vertical' interoperability between in-house components; or (b) 'lateral' interoperability across components developed for different enterprise workflows.

- 'Vertical' versus 'Lateral' Interoperability:
  - Vertical interoperability refers to the ability of components and (sub)systems to work with other components and (sub)systems within organization-specific workflows, where components are only functional if they can integrate into the rest of the workflow.
  - Lateral interoperability is concerned with the (more challenging) problem of developing components and systems with a more long-term outlook to facilitate interoperability with other (currently unknown) systems and mitigate potential difficulties upfront.

Figure 1 illuminates the contrast between vertical and lateral interoperability with an (over)simplified schematic. Real industrial workflows are often much more complicated. The key distinction to emphasize is that in vertical integration scenarios, representations, algorithms, and common properties are known and controlled by a single party. The success of lateral interoperability scenarios, on the other hand, hinges on the ability for two or more parties to find the common properties between different representations and algorithms, to which we shall return in Section 2.

## 1.2 Scope and Outline

This guide aims to formulate and study semantic interoperability in the general context of computational design and manufacturing, and discuss common approaches and alternative solutions for (vertical as well as lateral) interoperability. The goal is to provide a systematic framework to formally and precisely define the central concepts, starting from a high-level abstraction of models and representations of data and processes. This, in turn, will serve as a foundation to uniformly characterize practical interoperability scenarios and challenges.

The purpose of this document is not to present an exhaustive list of interoperability problems or solutions, which would require extensive domain knowledge and application-specific strategies. Rather, it is to provide a set of guidelines for domain experts to characterize and/or measure interoperability (or lack thereof) in different use-case scenarios, to understand commonalities between interoperability challenges, and to systematically approach developing interoperability solutions.

After briefly and informally overviewing the common approaches to and existing challenges in interoperability in the remainder of this section, the basic definitions of interchangeability, interoperability, and integration are given in Section 2. We advocate a paradigm shift from the unattainable ideal of 'model-based' equivalence of representations and/or algorithms to a practical prospect of 'property-based' equivalence (i.e., interchangeability). We discuss the importance of invariance of the said properties when establishing a correspondence between systems (i.e., interoperability) and

when assembling them into compound systems (i.e., integration). Particular attention is paid to the role of geometric properties as a surrogate to various other structural, physical, and behavioral properties that depend to a substantial extent on geometric specifications. The anatomy of an interoperability scenario, a classification of possible settings that one deals with, and a systematic methodology for verification of interoperability are also presented in Section 2, to be used as a recipe to systematically generate and examine several use-case examples in computer-aided design, analysis, manufacturing (CAD/CAE/CAM) in the subsequent sections.

In Section 3, geometric interoperability (e.g., between CAD systems) is delved into, examining the six major mechanisms that continue to be used for CAD-CAD data exchange, aiming at preserving shape (e.g., combinatorial, topological, geometric, differential, and integral) properties. Each scenario is analyzed in terms of the properties, if/how they are carried over from one system to another, and how their invariance after the exchange can be verified. The common advantages, drawbacks, and best practices are discussed from a property-based perspective.

In Section 4, we briefly discuss how the role of geometric surrogate properties allows one to extend these mechanisms to CAD-CAx interoperability and integration, in spite of the indisputable limitations, with use-case scenarios from CAD-CAE and CAD-CAM integration.

In Section 4.5, we briefly discuss directions for extending the framework to other interoperability scenarios such as multi-scale material modeling and multi-level system modeling.

Section 5 concludes the document. Appendix A provides a recipe of action items for the reader to setup their own interoperability scenarios.

### 1.3 State-of-the-Art

There have been (and continue to be) numerous standardization and unification efforts to address interoperability challenges in various contexts and applications. The existing approaches can be divided into four broad categories that are based on

- exchanging generic specifications of procedural (e.g., parametric, generative, or constructive) or declarative (e.g., constraint-based) models, assuming consistent expressions in different systems via compact symbolic structures;
- exchanging and converting fully instantiated representations of (presumably) agreed-upon common models, using either pairwise system-to-system or standardized system-to-neutral format translators via importing/exporting files;
- packaging functionalities (i.e., services) of some systems into stand-alone libraries accessible to client systems via application programmer's interfaces (API);
- standardizing on 'queries' defined with respect to common external semantics, which are exchanged between systems via interactive send/receive protocols;

or a combination of them. The first two are more broadly characterized as 'data-centric' methods, while the last two are loosely grouped together as 'query-based' approaches. In particular, exchanging procedural specifications of generic models and fully instantiated representations of non-generic (but consistent) models are both viewed as data-centric communication (e.g., via file export/import). The key distinction is that the former takes semantics for granted while the latter additionally requires representation conversion to make the semantic connection. On the other hand, API-based client-and-server or peer-to-peer information exchange can be viewed as query-based communication (e.g., via API function calls), the key distinction being pertinent to whether the semantics are specified by the developers and presumed by the clients, or if they both appeal to an standard external reference. However, the conceptual boundaries may not always be clear.

### 1.3.1 Data-Centric Approaches

**Generic Modeling: Circumventing Interoperability Problems?** A premier example of the generic modeling approach is the feature-based methodology for solid modeling [62, 27, 14, 9, 67], while similar approaches are abundant in various applications of computer-aided technologies (CAx) to scientific and engineering problems. In each specialized domain, mathematical models of physical phenomena are proposed, refined, and improved within expert communities as finite ontologies with (some degree of) consensus on the semantics of their building blocks (e.g., 'features')—i.e., what a 'slot', 'pocket', or 'hole' precisely mean and how they are instantiated for a given set of parameters in two different solid modelers, what composite 'laminates' are comprised of, how crystallographic 'unit cells' are characterized, and which parameters fully specify 'atoms' of nano-structures.

The ambitious goal of the generic modeling approach is to reduce the problem to one of merely syntactic interoperability, when consistent system-level interpretations (i.e., semantics) can be taken for granted. Although it may be a viable solution to vertical interoperability in relatively small systems over a short lifespan, by choosing and fixing generic models upfront and maintaining consistent semantics (formally or informally) across a single platform, it often becomes part of the problem rather than a solution for lateral interoperability. Achieving universal semantics, perfect standards, and consistent support for generic models in different systems is rendered increasingly unrealistic as design and manufacturing systems grow more complex and multifarious.

**Representation Conversions and Neutral File Formats.** Most modern design and manufacturing systems have developed around industry- or company-specific workflows, through which engineering information advance in a variety of application-specific data structures or neutral file formats. Success of such data-centric approaches hinges on the ability to perform representation conversions and format translations for all relevant aspects of the design and manufacturing information, forming what is often called a 'digital thread'.

In the data-centric approach to interoperability, it is often implicitly assumed that the data flowing through the digital thread is *informationally complete* at every stage, in a sense that it contains all the information required by every relevant downstream application, and its integrity remains intact throughout the pipeline. This implies a presumption of the existence of common abstract models, to which the two representations before and after conversion refer, either exactly or approximately, making them equivalent in some sense.

Unfortunately, the promise of informational completeness turns out to be largely a myth, even in the case of geometric information, where the notion appears to have a sound theoretical basis.[2] In practice, the incompatible assumptions often demand ad hoc heuristic recipes for salvaging the missing pieces of information, and manual domain-expert interventions break the digital thread, resulting in costly and error-prone workflows. Among other difficulties, successive back-and-forth data translations between even a single pair of representations from different systems with (even slightly) different semantics can lead to serious model degradation, especially in iterative workflows. This is primarily due to the fact that unless the two representations before and after conversion refer to precisely the same abstract model—which is rarely the case—it is extremely difficult to elicit an equivalence relation between them. In particular, the relation between two models described as "being approximately the same" is not transitive, and hence is not an equivalence. This poses a major challenge for interoperability as we will discuss in Section 2.2.2. In practice, it manifests as an accumulation of errors along the digital thread; namely, small deviations that are either negligible or repairable in a single conversion can lead to prohibitive discrepancies whose divergent behavior is not well-understood.[3]

[2] This assumption is more explicitly asserted for geometric information, thanks to the defining principles of solid modeling with a strong emphasis on informational completeness, when the crucial role of unambiguous geometric description was recognized for computerizing the numerically controlled (NC) machining in 1970s and 80s [59, 64].

[3] Anyone who has tried the amusing game of translating a simple paragraph on Google Translate™ back-and-forth between two languages from dramatically different linguistic families can relate to the intrinsic difficulties of a

A careful assessment of product data technologies [22] reflects that in reality, neutral formats consist of many different formats and variations, with substantial variations depending on limitations, assumptions, and interpretations of distinct commercial systems. Despite the monumental PDES/STEP effort that has been ongoing for several decades and has resulted in a body of standards for the exchange of shape data [2, 34], integrating shape (geometry and topology) representations from different CAD systems remains imperfect. According to various reports, failure rates ranging between 5–20% are not uncommon, well over 80% of companies are now using multiple CAD systems and formats,[4] and over 40% of professionals are now engaged in some interoperability tasks.[5]

In addition to the STEP standard(s), de facto specialized standards exist in different application domains, including IGES for curve and surface data [66], STL and (more recently popularized) 3MF for additive manufacturing (AM), OBJ for visualization, VRML for virtual reality (VR), FMI for co-simulation, CGNS for computational fluid dynamics (CFD) analysis, and more.

On the other hand, proprietary system-to-system translators provide by far the most effective interoperability solution between any pair of systems. However, such translators provide no guarantees and cannot account for incompatible assumptions and representational power of different systems. Customized translation also leads to an explosion of special purpose translators—growing at least quadratically with the number of systems—which severely inhibits the ability to systematically add, extend, or compose new representations and services to keep up with the evolving needs.

The above challenges notwithstanding, data-centric interoperability has become the industry standard for integrating heterogeneous systems due to its ability to completely decouple the operation of the individual systems. However, until these challenges are overcome, data-centric approaches will continue to suffer from robustness challenges and lack of guarantees. This in turns implies that data centric approaches do not support full automation, are not likely to scale, and may not support real-time and interactive applications.

### 1.3.2 Query-Based Approaches

**Libraries and Application Programmer's Interfaces (API).** The widespread demand for integration of different computational design and manufacturing technologies have led to componentization and packaging of different layers into stand-alone libraries that are accessible via application programmer's interfaces (APIs). For example, there are numerous APIs that support packaging of selective solid modeling capabilities and components from stand-alone libraries, including the popular kernels such as Parasolid, ACIS, OpenNURBS, and OpenCASCADE (based on B-reps), Hyperfun (based on F-reps), PADL-2 (based on CSG), Meshmixer (based on meshes), and OpenVDB/GVDB (based on voxels). However, it is important to note that such APIs are usually fine-tuned to the representation scheme and algorithmic infrastructure they encapsulate. As such, they are not interchangeable in the sense that replacing one component with another is usually difficult, if not impossible. Moreover, APIs are not necessarily interoperable when they are based on incompatible mathematical models and representations.

Notably, in the context of geometric modeling, a significant effort (called Djinn [5]) undertook the challenge of designing a representation-independent API for solid modeling based on canonical decompositions of the shape that underlies most finite set-theoretic representations [63]. The results of the effort remained largely academic because such decompositions are nontrivial to compute, leading to excessive fragmentation of geometric information, and are rarely used in practice.

The major difficulty with API-based interoperability is that it is restricted by the functionalities that the developers choose to publish for the clients to access through API services, and more importantly, it is reliant on typically non-standard semantics that are predominantly determined by the evolutionary development process of the libraries. Depending on how meticulously the API ser-

translation-based approach to semantic interoperability.

[4] Based on a 2010 survey of 269 companies, by the Aberdeen Group, www.aberdeen.com.

[5] Based on a 2010 "Collaboration & Interoperability" report by LongView Advisors www.longviewadvisors.com.

vices are documented and their semantics are published—e.g., in terms of their pre-/post-conditions, internal architecture, exception handling, and alike—proper function at the client's end is heavily dependent on a set of implicit *assumptions* about the server's internal working and interface properties that are partially untrue or subject to change in subsequent versions.

**Queries Standardized via Common External Semantics.** An alternative approach advocated in [29, 30] is to standardize on basic queries, which are computable functions or predicates that encapsulate and abstract details of individual representations. It is radically different from the traditional data-centric approach in that components and (sub)systems communicate not in terms of (supposedly) informationally complete representations and file formats, but in terms of standardized queries that provide a means for partial (i.e., incomplete) on-demand information exchange across communicating components or systems. The query-based approach may also be considered an extension of using APIs to communicate between different systems, the key distinction being that queries are defined by common formal specifications as opposed to system-specific implementation of subroutines that developers choose to expose for other applications.

The query-based approach enables adaptive and interactive communication and avoids global representation conversions and format translations altogether, along with their susceptibility to error accumulations. It is disruptive conceptually and technologically, and has proven effective in several scenarios, including seamless integration of solid modeling with structural simulation [20], manufacturability analysis for 3D printed parts [49], and computational modeling of material structures [39]. However, in spite of the promising prospects, the query-based approach is at a stage of infancy even in the better-understood realm of geometric information. At this time, its power, formal properties, and practical limitations remain to be established and validated in the broader context of computational design and manufacturing.

## 1.4 A Common Framework

Given the great diversity of representation schemes (and format converters), stand-alone systems (hardware and software), and APIs for computational design and manufacturing, the need for interchanging, sharing, and combining digital product information from different sources has become greater than ever. The two approaches—data-centric and query-based—represent two extreme methods of interoperability that have dual properties. The data-centric approach treats information as a representation of the system's state or history, while the query-based approach focuses on the process by which such information is measured, interpreted, utilized, and transmitted. It is conceivable that other approaches to interoperability are possible, but most of them are likely to be a combination of the aforementioned flavors.

The purpose of this document is not to advocate for or against a particular approach compared to the others, nor is it to provide interoperability solutions of either kind. Rather, it aims to provide a systematic framework to characterize and deploy the data-centric and query-based approaches in terms of their formal properties, while maximizing advantages and documenting limitations of the chosen approach to interoperability. This requires a general formulation of interoperability in the context of design and manufacturing applications; this is the subject of the next section.

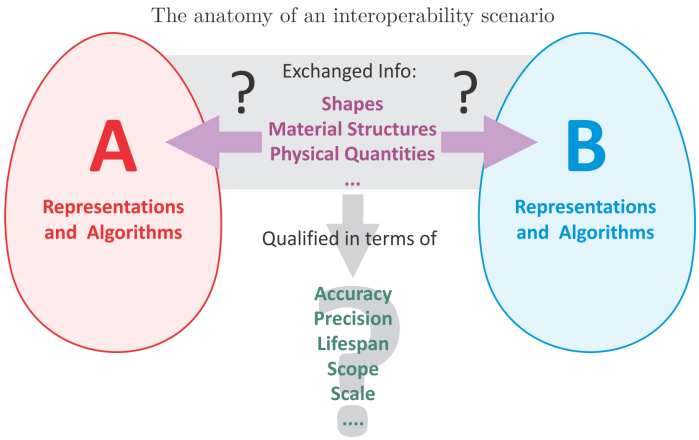



Figure 2: Semantic interoperability of computational design systems relies on qualifying the integrity of the exchanged information by considering accuracy, precision, lifespan, scope, scale, etc. These qualifications should be quantified with respect to a common external reference.

## 2 Definitions and Formulation

It is tempting to assume that the interoperability principles practiced in VLSI electronics, software engineering, programming languages, and web development are applicable with minor modifications to computational design of engineering (e.g., electromechanical, thermofluidic, and generally multi-physical) systems. An important distinction of engineering information is that it cannot be completely characterized by ontology and type, making it challenging to effectuate a 'plug-and-play' (PnP) development style. All such information should be further qualified with the recognition that computational models in design and manufacturing

- rely on *heterogeneous* information types (e.g., shape, material, physics, etc.) that come with a rich body of domain-specific semantics, assumptions, and even contradictions;
- are inherently *inexact*, requiring them to be accompanied with specifications of accuracy, precision, resolution, tolerances, etc.;
- are structurally *complex*, in a sense that they require description at multiple *scales* (in space and time) and with multiple *views* (of form and function);
- have dynamic *scope* and *lifespan*, affected by product life cycle and technological innovation.

It is not surprising that interoperability between computational systems in terms of such information-intensive, multi-scale, multi-view data structures and inevitably fallible algorithms cannot rely on an idealistic model of *equality*—e.g., a naïve presupposition that a pair of systems, developed for different purposes and based on different worldviews, can have an identical (i.e., truly equal) concept of a product's model. Nevertheless, intuitively it is clear that inteoperability of systems need a notion of *equivalence*, at one level of granularity or another, upon which abstract semantic commonalities of systems may be based. This leads us to the following fundamental principle:

**Principle of Interchangeability**

Interoperability between distinct systems (possibly rooted in different semantics) requires a notion of equivalence between informational elements of these systems. This notion of equivalence is hereafter called *interchangeability*.

In other words, the two-way horizontal arrow that illustrates some form of information exchange in Fig. 2—regardless of its operational facets, e.g., data-centric or query-based, open-loop or reciprocal, sequential or in-parallel, etc.—does not provide a complete picture of interoperability. The interchangeability of the exchanged information and their interpretations at the opposite ends of the arrow must be qualified with respect to the properties that need to be preserved throughout the information exchange, in spite of the differences in terms of application-specific constraints on the shape (geometry and topology), material structure, physical quantities, manufacturing parameters, and other information. The question is of specifying or characterizing the *invariants* in this information exchange that define the essence of interchangeability.

On the other hand, in observance of possible semantic differences between the systems—none of which necessarily provides the "ground truth" based on which to qualify the data—the interchangeability of the information residing at the opposite ends of the horizontal arrow in Fig. 2 must be qualified with respect to *common semantics* that are agreed-upon by the two systems. The inherent inexactness of representations and algorithms makes this task particularly challenging because the common semantics must account for some or all of the following: precision of numerical data and computations, accuracy of algorithms, tolerances of models and representations, as well as their resolution and scale.

Semantic interoperability of computational systems is defined in terms of the *invariants* of the information exchange, that should be precisely specified with respect to *common semantics*.

The central premise of this document is that any practically realistic interoperability scenario must explicitly specify and document the aspects of the information exchange that must remain invariant. These "aspects" (hereafter called 'properties') must be made sense of by appealing to a common semantic reference. Accordingly, any attempt to certify the interoperability of the two systems must first verify the interchangeability of exchanged information, which, in turn, amounts to inverstigating if *the specified properties are preserved throughout the exchange*.

The invariants of an interoperability scenario can be defined as set of concrete *properties* that must be preserved throughout the information exchange, regardless of the operational means.

As illustrated by Fig. 3, computing and interpreting the properties of the information carried by each system with respect to common semantics can be conceptualized by a pair of mappings from the two systems into a space of properties. If 'Alice' $\mathsf{A}$ and 'Bob' $\mathsf{B}$ are a pair of systems whose interoperability (or lack thereof) is being studied, the property space $\mathsf{P}$ is a common data type that is recognized by both systems. Each system takes the responsibility of correctly computing the properties on its internal elements (with possibly hidden details) in compliance with the external semantics—abstracted as the so-called 'property functions' $\mathsf{A} \to \mathsf{P}$ and $\mathsf{B} \to \mathsf{P}$ in Fig. 3. The information exchange itself, regardless of its operational means, can be abstracted as mappings as well, called 'interoperability maps' $\mathsf{A} \rightleftarrows \mathsf{B}$ in Fig. 3.

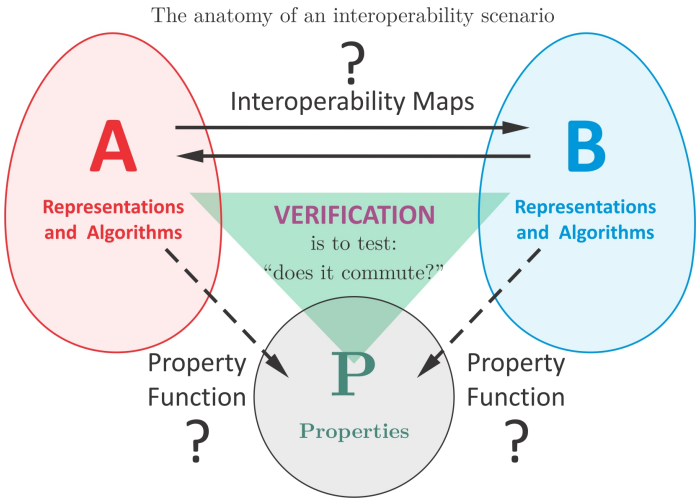



Figure 3: Computing the properties (for qualifying interchangeability) and the information exchange (for enabling interoperability) are conceptualized as mathematical mappings.

In Section 2.1, we make the above notions precise at a high-level of abstraction, accompanied with a few simple concrete examples. The goal is to present the common ingredients of interoperability scenarios in a clear, widely-accessible, and self-sufficient language.

In Section 2.2 we will elaborate on the notion of invariants and present a few central concepts including *transitivity* (of interchangeability), *external addressability* (of common references), the role of geometry as a *surrogate* to non-geometric properties, and *granularity* (of equivalence classes).

In Section 2.3, we classify interoperability problems into four major classes depending on the known/unknown arrows on the diagram of Fig. 3. The developed abstractions and language are used to set up interoperability scenarios, examples of which are studied in Section 3.

## 2.1 Basic Definitions

Before discussing their interoperability, we must first clarify what we mean by 'systems':

**Computational Systems** (Fig. 4)

A computational 'system' $\mathsf{S}$ is a computer implementation of an abstract modeling space C:*

- $\mathrm{C} = \langle \mathrm{M_C}; \mathrm{F_C} \rangle$ denotes a collection of mathematical 'models' (denoted $\mathrm{M_C}$) and mathematical 'functions' (denoted $\mathrm{F_C}$) acting on those models, to abstract and explain physical states and processes, respectively.
- $\mathsf{S} = \langle \mathsf{M_S}; \mathsf{F_S} \rangle$ denotes a collection of computer 'representations' (denoted $\mathsf{M_S}$) and computational 'algorithms' (denoted $\mathsf{F_S}$) computing on those representations, which implement the aforementioned abstractions.

Hereafter, we shall refer to a pair of generic systems, whose interoperability is of interest, as 'Alice' $\mathsf{A} = \langle \mathsf{M_A}; \mathsf{F_A} \rangle$ and 'Bob' $\mathsf{B} = \langle \mathsf{M_B}; \mathsf{F_B} \rangle$, color-coded as such in all notations and figures.

*Hereafter, entities which belong to the common external reference are denoted via Roman (serif) symbols, whereas entities which belong to system-specific computational spaces are denoted via Gothic (sans serif) symbols.

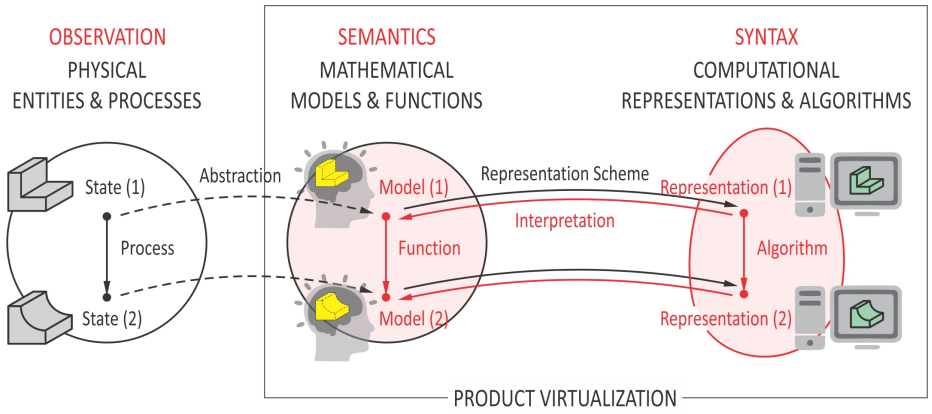



Figure 4: The abstraction paradigm common to many modeling activities (recreated from [72]).

Thus a computational system can be viewed as a syntactic structure factory interpreted by a conceptual map (i.e., 'semantics') $\gamma_{\mathsf{S}} : \mathsf{S} \to \mathrm{C}$ which maps representations to models, and accordingly, maps algorithms to functions. This view is consistent with how representations are traditionally conceived and developed in computer-aided design and manufacturing (Voelcker and Requicha [72])—in which lexicon the inverse $\gamma_{\mathsf{S}}^{-1} : \mathrm{C} \to \mathsf{S}$ is referred to as the 'representation scheme' [58] (Fig. 4).[6]

**Units of Information: Models and Representations** (Fig. 10 (b))

For our purposes, models stand for mathematical abstractions of 'states', physical or virtual, represented in each system via finite symbol structures. They may refer to logical information, combinatorial networks, shape (geometry and topology), lattice structures, continuum space-time distributions (i.e., tensor fields), discrete forms (i.e., co-chains), configurations (i.e., motions), tolerance specifications (e.g., GD&T datums), process parameters, and meta-data.

**Units of Computation: Functions and Algorithms** (Fig. 10 (c))

On the other hand, functions are conceptualizations of 'processes', physical or virtual, ranging from computing local or global numerical properties (e.g., evaluating differential and integral properties) to transformations of the models (e.g., computing the medial axis transform), combinations of two or more models into different models (i.e., performing Boolean or Minkowski operations), and so on, implemented in each system via computable algorithms or subroutines. They form the building blocks for computational tasks throughout the digital thread, including data acquisition, geometric modeling, shape synthesis, physical analysis, manufacture planning, optimization, inspection, documentation, and archival.

[6]The modeling itself is also viewed as another conceptual map from a physical referent (e.g., object or process) to the modeling space, giving rise to the information triplet ⟨object, model, representation⟩ [56, 58]. Similar conceptualizations have been proposed by others in different contexts of computer science and engineering, including Scott and Strachey's denotational semantics for programming languages [60], Denning's model of representation transformations [17], and Cook's data abstraction paradigm for programming [15].

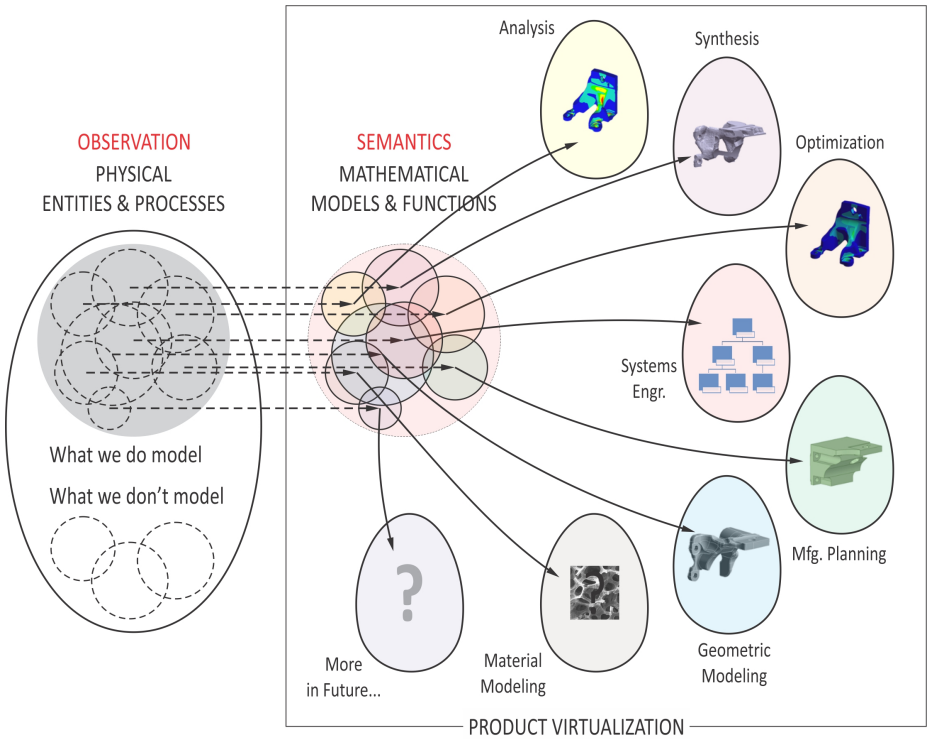



Figure 5: The more realistic picture of computer-aided technologies (CAx), each focusing on particular aspects of modeling, computing, and reasoning. Geometry plays a central role in abstraction as well as communication between different CAx components—compare to Fig. 4.

**Computer-Aided Technologies.** Although the above definition is fairly general—and so are the subsequent developments—we are primarily interested in computer-aided technologies (CAx), including but not limited to

- CAD systems for modeling geometric shapes (geometry and topology) with or without annotations (e.g., material structures, GD&T specs, etc.);
- CAE systems for analyzing geometric models for simulating physical behavior (e.g., structural, thermal, aerodynamic, etc.);
- CAM systems for manufacturability analysis and fabrication process planning;[7]
- design exploration, shape synthesis, and topology optimization systems;
- multi-scale material modeling languages;
- system modeling languages; and others.

These systems focus on different aspects of a product's digital model—i.e., modeling, computing, and reasoning from a different perspective for a specific application—as depicted by Fig. 5.

[7]The term computer-aided manufacturing (CAM) is often reserved for tool-path generation and machine-level control activities, and is treated differently from computer-aided process planning (CAPP) used for high-level planning. Here, we use "CAM" in CAD-CAM integration to broadly refer to all computer-assisted manufacturing activities.

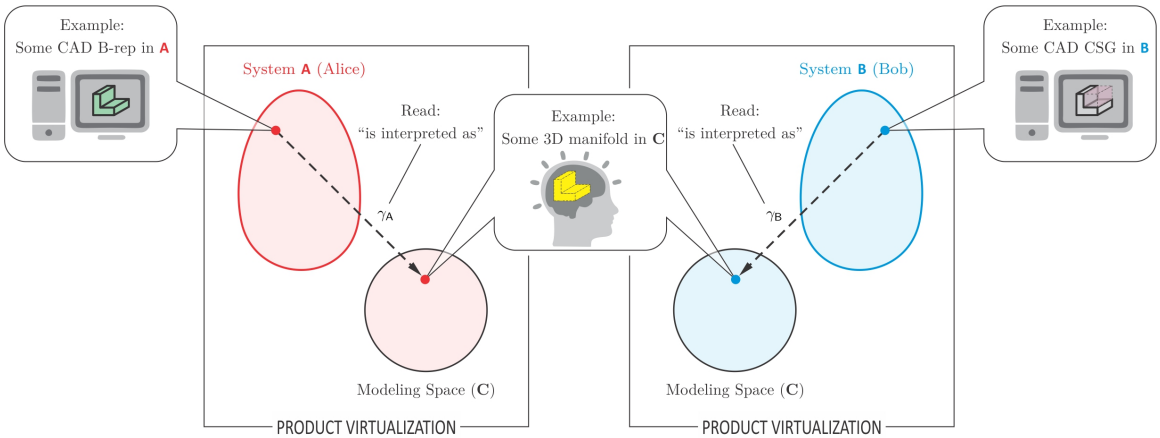



Figure 6: The same 3D solid model represented by B-rep and CSG schemes.

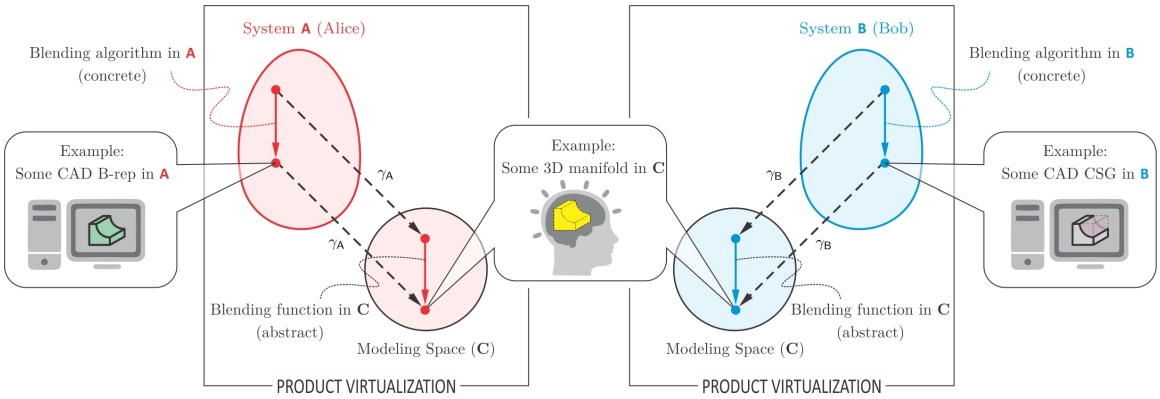



Figure 7: The same CAD operation implemented on B-rep and CSG schemes.

**Example 2.1.** Consider two traditional CAD software/libraries $\mathsf{A} = \langle \mathsf{M}_\mathsf{A}; \mathsf{F}_\mathsf{A} \rangle$ and $\mathsf{B} = \langle \mathsf{M}_\mathsf{B}; \mathsf{F}_\mathsf{B} \rangle$. The representation spaces $\mathsf{M}_\mathsf{A}$ and $\mathsf{M}_\mathsf{B}$ are the space of all valid data structures that each system can instantiate following the rules of a specific representation scheme; e.g., CSG, B-reps, F-reps, voxel maps, point clouds, etc. [58]. These representations are interpreted via the systems' $\gamma-$maps to some universe of mathematical objects $\mathrm{M_C}$ that exist independently of the representation scheme, and are (presumably) common between different CAD systems (Fig. 6). Accordingly, $\mathsf{F}_\mathsf{A}$ and $\mathsf{F}_\mathsf{B}$ are the space of different algorithms programmed to operate on those data structure; e.g., Boolean operations, freeform surface editions, offsets/blends, integral property computations, etc. These algorithms are interpreted via the systems' $\gamma-$maps—by extending the interpretation to their inputs/outputs—to the mathematical functions acting on the aforementioned abstract models (Fig. 7).

Thus it is conceivable that representations and/or algorithm of $\mathsf{A} = \langle \mathsf{M}_\mathsf{A}; \mathsf{F}_\mathsf{A} \rangle$ and $\mathsf{B} = \langle \mathsf{M}_\mathsf{B}; \mathsf{F}_\mathsf{B} \rangle$ can be deemed interchangeable with respect to the common models and/or function in $\mathrm{C} = \langle \mathrm{M_C}; \mathrm{F_C} \rangle$ as illustrated by the diagram in 7. Such an idealistic concept of model-based interchangeability—often called 'consistency' in traditional solid modeling literature [58, 57]—will break down as soon as the systems have different semantics, assumptions, constraints, representational precisions, algorithmic accuracies, etc.

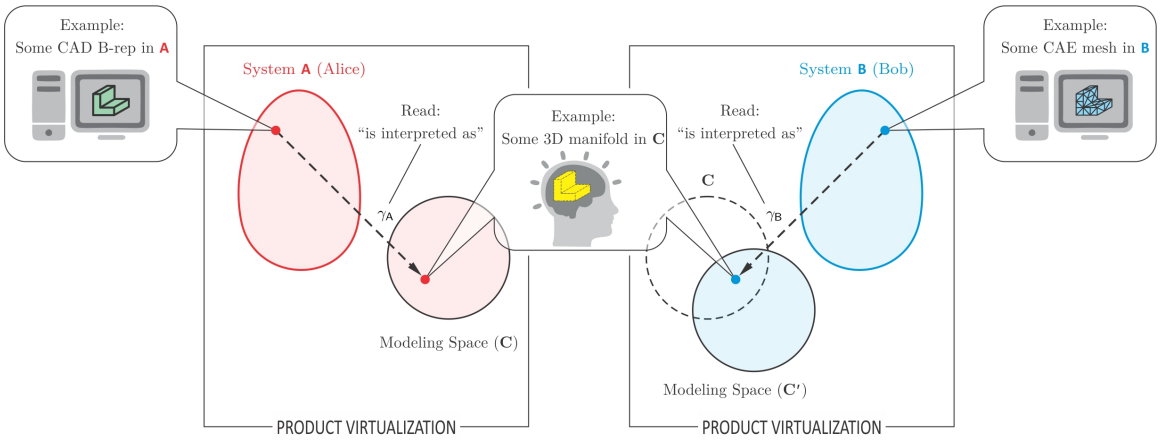



Figure 8: The same 3D solid model represented by B-rep and triangular mesh.

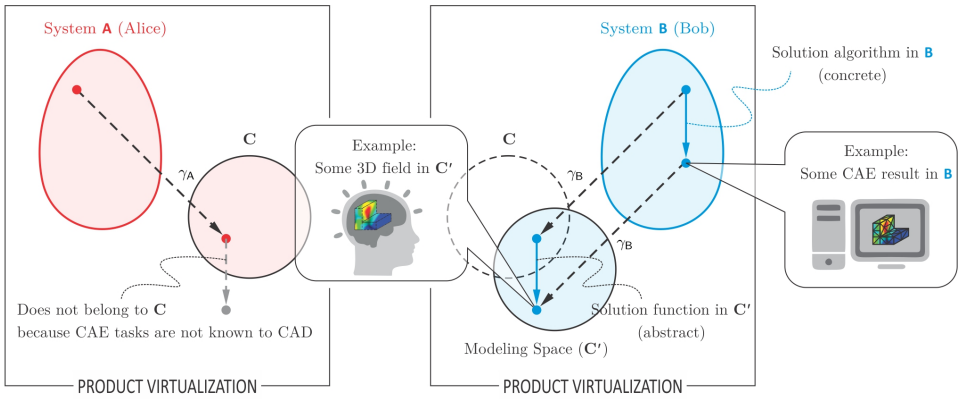



Figure 9: The CAD system has no interchangeable algorithm for CAE analysis.

**Example 2.2.** For a downstream CAx system $\mathsf{B}' = \langle \mathsf{M}_{\mathsf{B}'}; \mathsf{F}_{\mathsf{B}'} \rangle$ that draws upon the CAD model (e.g., from A in Example 2.1), models in $\mathrm{M}_{\mathrm{C}'}$ and their representations in $\mathsf{M}_{\mathsf{B}'}$ as well as functions in $\mathrm{F}_{\mathrm{C}'}$ and their algorithms in $\mathsf{F}_{\mathsf{B}'}$ can be substantially different than those of the CAD system. The shape (geometry and topology) information is still central to $\mathsf{M}'_{\mathsf{B}}$—sometimes using simplified models or partial information to the extent that is sufficient for the particular purpose. Additionally, other types of information may be represented on top of the geometric backbone—e.g., material structures physical quantities, process parameters, etc. We will see in Section 2.2.1 how this common backbone can be exploited to draw some form of interchangeability (i.e., equivalence) between unequal elements of the two systems.

Examples of $\mathsf{M}_{\mathsf{B}'}$ are simplified (e.g., polygonized and de-featured) mesh structures augmented with discretized physical fields for CAE, decomposed or sliced representations and cutter/nozzle trajectories for CAM, homogenized material models at different size scales, reticulated (i.e., lumped) networks for system modeling, etc. In addition, the CAx system offers task-specific algorithms $\mathsf{F}_{\mathsf{B}'}$ that implement new functionalities (not recognized by the CAD system) related to physical simulation, fabrication planning, material characterization, system dynamics modeling, etc.

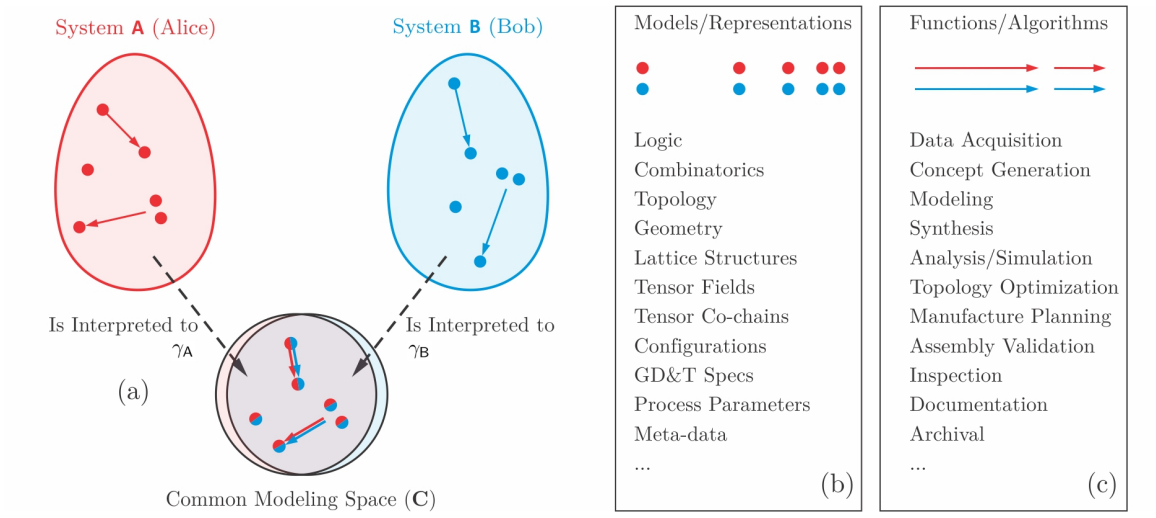



Figure 10: A schematic of systems A and B and their (presumably) common modeling space. The models/representations (bullets) are operated on by functions/algorithms (arrows).

### 2.1.1 Interchangeability

The simple (and quite universal) information model presented in Fig. 4 is helpful in conceptualizing computational modeling efforts. In particular, it provides a first shot at defining a 'model-based' interchangeability (i.e., 'consistency' as defined by Requicha [58]):

**Model-Based Interchangeability** (Fig. 10)

Given systems $\mathsf{A} = \langle \mathsf{M_A}; \mathsf{F_A} \rangle$ and $\mathsf{B} = \langle \mathsf{M_B}; \mathsf{F_B} \rangle$ and a common modeling space $\mathrm{C} = \langle \mathrm{M_C}; \mathrm{F_C} \rangle$ to which they both map their elements via $\gamma_\mathsf{A} : \mathsf{A} \to \mathrm{C}$ and $\gamma_\mathsf{B} : \mathsf{B} \to \mathrm{C}$, respectively:

- A pair of representations in $\mathsf{M_A}$ and $\mathsf{M_B}$ (denoted $\mathsf{m_A} \in \mathsf{M_A}$ and $\mathsf{m_B} \in \mathsf{M_B}$) are called model-based interchangeable (or '$\gamma-$interchangeable' for short, denoted $\mathsf{m_A} \overset{\gamma}{\equiv} \mathsf{m_B}$) if they both represent the *exact* same model, i.e., $\gamma_\mathsf{A}(\mathsf{m_A}) = \gamma_\mathsf{B}(\mathsf{m_B}) = \mathrm{m_C}$ (where $\mathrm{m_C} \in \mathrm{M_C}$).
- A pair of algorithms $\mathsf{f_A} \in \mathsf{F_A}$ and $\mathsf{f_B} \in \mathsf{F_B}$ are similarly defined $\gamma-$interchangeable if they both implement the same function, i.e., $\mathsf{f_A} \overset{\gamma}{\equiv} \mathsf{f_B}$ if $\gamma_\mathsf{A}(\mathsf{f_A}) = \gamma_\mathsf{B}(\mathsf{f_B}) = \mathrm{f_C}$ (where $\mathrm{f_C} \in \mathrm{F_C}$).

Accordingly, the systems A and B have 'total' $\gamma-$interchangeability with respect to C if they both cover the entirety of C by their interchangeable elements. The interchangeability is called 'partial' if it is total over a proper subset of semantic space $\mathrm{C}' \subset \mathrm{C}$.

**Example 2.3.** For CAD systems, the space of mathematical models/functions C is universal—e.g., widely accepted choices (for $\mathrm{M_C}$) are orientable manifold cell complexes or semianalytic 'r-sets' in 3D [56], along with set-theoretic and topological operations on them (for $\mathrm{F_C}$). Let A and B be two arbitrary CAD kernels implementing that space—e.g., Parasolid and ACIS, or pick your own from examples given in Section 1.3.2. They may use different representations schemes (e.g. different flavors of B-reps, meshes, CSG, features, or combinations of them) and different algorithms optimized for each scheme. Nevertheless, they are deemed $\gamma-$interchangeable as long as their mathematical $\gamma-$interpretations are consistent.

Even for the example of CAD-CAD interchangeability given above, where semantics are fairly universal, there are at least two main problems with the model-based approach in practice; namely,

- model-based interchangeability *is never total*, i.e., it is rarely the case for two systems based on different representation schemes, to have the same expressive power, i.e., to be able to represent the same extent of models and compute the same extent of functions residing in a superset of their modeling spaces C. In reality, different representations are suitable for different tasks, explaining their existential diversity in the first place.

- model-based interchangeability *is never exact*, i.e., even for the representations and/or algorithms of the same nature (e.g., B-rep in A versus B-rep in B) where one would expect perfect semantic consistency in an ideal world, exact interchangeability is challenged due to discrepancies in representation precision semantics and/or algorithmic accuracies and guarantees.[8]

Thus one has to speak of *partial* and *inexact* interchangeability even in the familiar case of CAD-CAD interchangeability with seemingly consistent and universally agreed-upon semantics.

More importantly, although Fig. 4 was a realistic picture in the not-so-distant past when there were only a handful of academic or industrial implementations per task, it does not suffice to reflect the current reality, which looks more like Fig. 5. Today, computer-aided technologies (CAx) have grown ever so diverse in computational tasks and representation/algorithmic schemes per subtask.[9] They are increasingly more focused on different aspect of the engineering information models and more specialized on near-optimal treatment of narrower (but deeper) areas of modeling, simulation, planning, optimization, etc.

Nevertheless, these different models/representations and functions/algorithms for different CAx applications ought to have *something in common*, in order to make interchangeability meaningful, and to enable interoperability by virtue of preserving those common aspects. We generically refer to those commonalities as 'invariant properties' (or 'invariants' for short). These invariants could be topological properties (e.g. homology or homotopy type), metric properties (e.g., Hausdorff distance from a reference set), differential properties (e.g. continuity or smoothness), integral properties, statistical descriptors, and so on. The invariants could be also defined experimentally or empirically, for example by prescribing the expected behavior of representations and/or algorithms with respect to reference data sets or test cases (e.g., membership test against a large number of points).

The semantics for interchangeability are given as a space of *properties* that are commonly stipulated by both systems. Interchangeability of elements is certified by matching properties, and interoperability of systems hinges on preservation of properties upon information exchange.

The space of properties is hereafter denoted via $\mathrm{P} := \langle \mathrm{M_P}; \mathrm{F_P} \rangle$ which includes models of properties (denoted $\mathrm{M_P}$) and functional relationships between them (denoted $\mathrm{F_P}$). If a system $\mathsf{S} = \langle \mathsf{M_S}; \mathsf{F_S} \rangle$ intends to interoperate with other systems with respect to a subset of these properties, it must come with a 'property function' $\mu_\mathsf{S} : \mathsf{S} \to \mathrm{P}$ that maps every internal computational element to an external property. The property functions must be well-defined and computable, at least in principle. When the property is implemented using the authoring system's own internal algorithms and published to its client systems, they must be able to recognize the same type of properties. This leads to the more general notion of 'property-based' interchangeability:

[8] There are many examples: choices of precision (e.g., $10^{-6}$ vs. $10^{-12}$), error quantifiers (e.g., absolute vs. relative), truncation rules (e.g., round half-up vs. half-down), number system (e.g., binary vs. decimal basis), floating-point arithmetics (e.g., single- vs. double-precision), and many others.

[9] One quick look at Wikipedia's incomplete list of the famous CAx vendors is enough to grasp the scale of the expected interoperability challenges: en.wikipedia.org/wiki/List_of_CAx_companies.

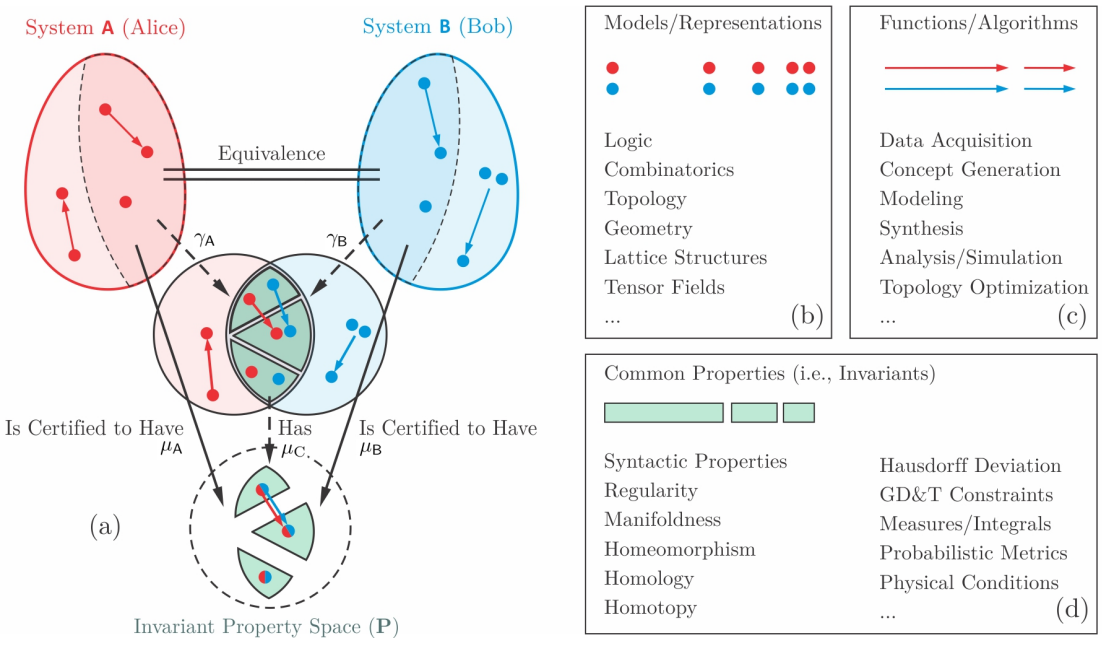



Figure 11: A schematic of systems A and B and their common property space. To obtain the latter, models/representations (bullets) and functions/algorithms (arrows) are grouped into equivalence classes (wedges) based on the properties they share, imposing a partitioning on the modeling space.

**Property-Based Interchangeability** (Fig. 11)

Given systems $\mathsf{A} = \langle \mathsf{M_A}; \mathsf{F_A} \rangle$ and $\mathsf{B} = \langle \mathsf{M_B}; \mathsf{F_B} \rangle$ and a common property space $\mathsf{P} = \langle \mathsf{M_P}; \mathsf{F_P} \rangle$ to which they both map their elements via $\mu_\mathsf{A} : \mathsf{A} \to \mathsf{P}$ and $\mu_\mathsf{B} : \mathsf{B} \to \mathsf{P}$, respectively:

- A pair of representations in $\mathsf{M_A}$ and $\mathsf{M_B}$ (denoted $\mathsf{m_A} \in \mathsf{M_A}$ and $\mathsf{m_B} \in \mathsf{M_B}$) are called property-based interchangeable (or '$\mu-$interchangeable' for short, denoted $\mathsf{m_A} \overset{\mu}{\equiv} \mathsf{m_B}$) if they both possess the same property, i.e., $\mu_\mathsf{A}(\mathsf{m_A}) = \mu_\mathsf{B}(\mathsf{m_B}) = \mathrm{m_P}$ (where $\mathrm{m_P} \in \mathrm{M_P}$).
- A pair of algorithms $\mathsf{f_A} \in \mathsf{F_A}$ and $\mathsf{f_B} \in \mathsf{F_B}$ are similarly defined $\mu-$interchangeable if they both implement the same function, i.e., $\mathsf{f_A} \overset{\mu}{\equiv} \mathsf{f_B}$ if $\mu_\mathsf{A}(\mathsf{f_A}) = \mu_\mathsf{B}(\mathsf{f_B}) = \mathrm{f_P}$ (where $\mathrm{f_P} \in \mathrm{F_P}$).

Accordingly, the systems A and B have 'total' $\mu-$interchangeability with respect to P if they both cover the entirety of P by their interchangeable elements. The interchangeability is called 'partial' if it is total over a proper subset of property space $\mathrm{P}' \subset \mathrm{P}$.

**Properties of Interchangeability.** It might appear at a first glance that the distinction between the modeling space $\mathrm{C} = \langle \mathrm{M_C}; \mathrm{F_C} \rangle$ (with semantic map $\gamma_\mathsf{S} : \mathsf{S} \to \mathrm{C}$) and property space $\mathrm{P} = \langle \mathrm{M_P}; \mathrm{F_P} \rangle$ (with property map $\mu_\mathsf{S} : \mathsf{S} \to \mathrm{P}$) is merely an artificial one. In fact, if models are viewed as what captures all computable properties, model-based interchangeability is to claim property-based interchangeability for *every* possible property, which is unrealistic.

Figure 11 illustrates the differences between property-based $\mu-$interchangeability and model-based $\gamma-$interchangeability. Each property in P can be viewed as a common aspect of a collection of models in C, forming an *equivalence class* in C (hereafter called an 'interchangeability class') illustrated via green wedges in Fig. 11. In other words, the $\mathrm{P}-$space corresponds to a partitioning

of the C−space such that elements that are "similar enough" by virtue of the properties we care about are grouped together.[10] Their differences are deemed irrelevant.

**Interchangeability Certificates: Invariant Properties** (Fig. 11 (d))

The computable properties that can be used to certify interchangeability in computational design and manufacturing include syntactic and semantic properties, the latter including but not limited to topological properties such as regularity, manifoldness, connectivity, homeomorphism, homology, homotopy type, etc.; geometric properties such as metric, affine, and differential properties (e.g., local curvatures) integral properties including Hausdorff measures (e.g., volume, surface area, curve length, and cardinality) and other 'lumped' properties, variation metrics including Hausdorff distance or GD&T deviations with respect to *external references* (elaborated in Section 2.2.2), shape descriptors such as Minkowski functionals, probabilistic measures such as multi-point correlation functions, physical properties such as satisfaction of governing conservation or constitutive equations, compatibility conditions, boundary/initial conditions, and many more.

**Example 2.4.** Let A and B be two CAD software systems using different representation semantics, precisions, accuracies, etc. When shape information are exchanged between them, the loss of information may be inevitable to some extent, depending on the system-specific constraints on both ends. Thus the representations before and after conversion may not be representative of the same exact model (i.e., $\gamma_{\mathsf{A}}(\mathsf{m}_{\mathsf{A}}) \neq \gamma_{\mathsf{B}}(\mathsf{m}_{\mathsf{B}})$). However, it is conceivable that one can come up with a set of properties whose invariance throughout the translation indicates "good enough" conservation of model integrity (i.e., $\mu_{\mathsf{A}}(\mathsf{m}_{\mathsf{A}}) = \mu_{\mathsf{B}}(\mathsf{m}_{\mathsf{B}})$). It can be a combination of combinatorial properties (e.g., B-rep cell complex homology), topological properties (e.g., homeomorphism to a reference), and geometric properties (e.g., being within tolerance zone of a reference).

**Example 2.5.** Let A be a CAD system and B be a CAE solver. Shape information is translated from the designed model with freeform surfaces and feature semantics to a polyhedral mesh (possibly approximated, simplified, or de-featured) before it is analyzed. This means that some information is lost in the transfer, and the CAE model is not the same as the CAD model (i.e., $\gamma_{\mathsf{A}}(\mathsf{m}_{\mathsf{A}}) \neq \gamma_{\mathsf{B}}(\mathsf{m}_{\mathsf{B}})$). Yet to guarantee correctness of the analysis results, certain combinatorial, topological, and geometric properties of the shape should be preserved after the meshing process (i.e., $\mu_{\mathsf{A}}(\mathsf{m}_{\mathsf{A}}) = \mu_{\mathsf{B}}(\mathsf{m}_{\mathsf{B}})$). The only properties that are considered relevant are those which can affect the CAE solution; for example, those that can ultimately affect analysis activities such as enforcing boundary conditions and integrating governing equations.

We should be standardizing on the *properties* that remain invariant when exchanging information across heterogeneous representations/algorithms implementing diverse models/functions. Standardizing on representations (e.g., neutral file formats) or algorithms (e.g., constructive procedures) is limited, as it counts on the illusory promise of model-based interchangeability.

[10] Technically, $\mathrm{P} \cong \mathrm{C}/\overset{\mu}{\equiv}$ is the quotient space of C modulo equivalence via $\mu_{\mathrm{C}} : \mathrm{C} \to \mathrm{P}$, where $\mu_{\mathsf{S}} := (\mu_{\mathrm{C}} \circ \gamma_{\mathsf{S}})$.

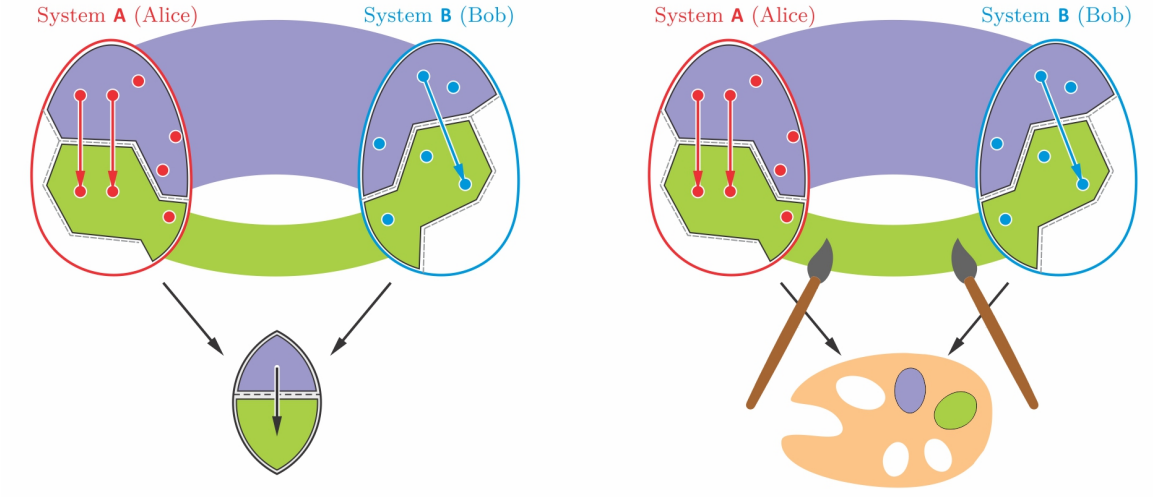



Figure 12: Interchangeability as a coloring: colors are properties and brushes are $\mu-$functions.

---

**Summary of Interchangeability.** To summarize Section 2.1.1, property-based interchangeability can be viewed as a consistent *coloring* of two systems (Fig. 12). If the color palette is the common semantic reference, each system has its own canvas (i.e., computational space of representations and algorithms) and its own brush (i.e., property functions), meaning that it is responsible to correctly paint its own computational elements (i.e., assign properties to them).

Interchangeability goes only as far as discovering to what extent the two systems have equivalent representations and/or algorithms. It establishes an implicit many-to-many association within and across equivalence classes in each system. It is implicit in the sense that one can test and decide (i.e. answer "is true or false?") the relation $\mathsf{m}_\mathsf{A} \overset{\mu}{\equiv} \mathsf{m}_\mathsf{B}$ for a given $\mathsf{m}_\mathsf{A} \in \mathsf{M}_\mathsf{A}$ and $\mathsf{m}_\mathsf{B} \in \mathsf{M}_\mathsf{B}$, by asking each system to separately compute the properties $\mu_\mathsf{A}(\mathsf{m}_\mathsf{A})$ and $\mu_\mathsf{B}(\mathsf{m}_\mathsf{B})$, then externally check if the two are equal $\mu_\mathsf{A}(\mathsf{m}_\mathsf{A}) = \mu_\mathsf{B}(\mathsf{m}_\mathsf{B})$. However, we do not yet have enough machinery to go from one element to another across the two systems while preserving the properties—which is the ultimate goal of interoperability. In particular:

1. How does the sending system (e.g., A) check if for any one of its elements (e.g., representations and algorithms) the receiving system (e.g., B) has any interchangeable elements at all?

2. If there exists one or more interchangeable elements in the receiving system's end, how are they chosen, i.e., what is/are the property-preserving mapping(s) between A and B?

In the next section, the notion of 'interoperability' is formally defined by the ability of the systems to do the above, i.e., by establishing a correspondence (called an 'interoperability map') between the systems that preserves the properties.

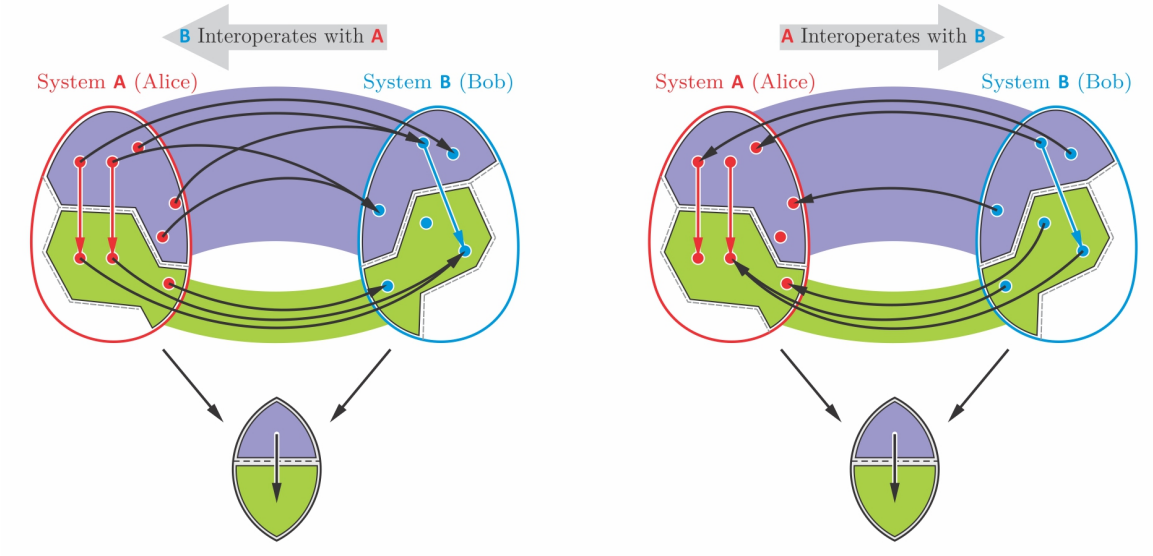



Figure 13: Interoperability as a correspondence that is faithful to the coloring in Fig. 12.

### 2.1.2 Interoperability

Informally, interoperable systems are not only capable of discovering the extent of their partial interchangeability, but also aware of a precise correspondence between their interchangeable elements.

**Property-Based Interoperability** (Fig. 13)

Let $\mathsf{A} = \langle \mathsf{M}_\mathsf{A}; \mathsf{F}_\mathsf{A} \rangle$ and $\mathsf{B} = \langle \mathsf{M}_\mathsf{B}; \mathsf{F}_\mathsf{B} \rangle$ be partially interchangeable systems with respect to $\mathsf{P} = \langle \mathrm{M}_\mathrm{P}; \mathrm{F}_\mathrm{P} \rangle$ to which they both map their elements via $\mu_\mathsf{A} : \mathsf{A} \to \mathsf{P}$ and $\mu_\mathsf{B} : \mathsf{B} \to \mathsf{P}$, respectively. A given map $\eta_{\mathsf{AB}} : \mathsf{A} \to \mathsf{B}$ is called an interoperability map (or '$\eta-$map' for short) if it preserves the properties, i.e., if its outputs and inputs are $\mu-$interchangeable.

- Given a pair of representations $\mathsf{m}_\mathsf{A} \in \mathsf{M}_\mathsf{A}$ and $\mathsf{m}_\mathsf{B} \in \mathsf{M}_\mathsf{B}$ associated via $\mathsf{m}_\mathsf{B} = \eta_{\mathsf{AB}}(\mathsf{m}_\mathsf{A})$, one has $\mathsf{m}_\mathsf{A} \overset{\mu}{\equiv} \mathsf{m}_\mathsf{B}$, i.e., $\mu_\mathsf{A}(\mathsf{m}_\mathsf{A}) = \mu_\mathsf{B}(\mathsf{m}_\mathsf{B})$.
- Accordingly, if an algorithm $\mathsf{f}_\mathsf{A} \in \mathsf{F}_\mathsf{A}$ is mapped to an algorithm $\mathsf{f}_\mathsf{B} \in \mathsf{F}_\mathsf{B}$ they must be interchangeable for all combinations of inputs and outputs.

Accordingly, the receiving system B is said to be 'interoperable' with the sending system A if a specific $\eta-$map is given from A to B (explicitly or implicitly).

**Properties of Interoperability.** A few points are worth emphasizing to interpret the above definition: First, unlike interchangeability, interoperability is a *non-symmetric* relation.

- Practically, this means that one system can correctly interpret and use the information received from the other system—where "correctly" means up to property-based interchangeability. But it may not be the case for the inverse information stream. For example, when a CAE system is said to interoperate with a CAD system, it must be able to understand shape data from CAD (e.g., exported to mesh or locally queried on-demand) but the CAD system may not be able to make sense of the deformations or other physical analysis results from CAE.

- Theoretically, this is because the $\eta-$map may not be bijective, thus not necessarily invertible. Multiple elements in the same equivalence class in $\mathsf{A}$ may collapse to a single element in $\mathsf{B}$, and other elements in the same equivalence class in $\mathsf{B}$ may not be covered by the map at all.

Secondly, it is not sufficient to merely know or assert the *existence* of an interoperability map. Interchangeability alone suffices to know that there are infinitely many $\eta-$maps that satisfy the above conditions.

- Practically, once we reject the idealistic expectation of model-based interchangeability, the $\eta-$map is inevitably lossy and is not unique. For example, when two CAD systems use dramatically different semantics, their lossy data exchange can be set up in various ways, e.g., preserving topological properties at the expense of $G^k-$continuity or vice versa, trading off combinatorial structure to preserve geometric accuracy, etc.

- Theoretically, this is captured by the fact that $\mu-$interchangeability is a many-to-many binary relation from $\mathsf{A}$ to $\mathsf{B}$. One can come up with various choices of outputs for a given input, leading to different many-to-one refinements (i.e., functions) for the $\eta-$map.

Thirdly, unlike the property functions that must be separately implemented and published by each individual system *explicitly*, the interoperability map is collectively specified by the two systems, either *explicitly* or *implicitly*—which is one of the fundamental differences between data-centric and query-based approaches, respectively.

- Practically, this means that the map need not be a one-time translation process (e.g., as in data-centric approaches). The requirement is that for every element in the sending system, the receiving system knows precisely, unambiguously, and deterministically what the corresponding element in the receiving system will be. The actual process could be a one-time data-centric conversion, interactive/reciprocal query-based reproduction, or a combination of the two.

- Theoretically, it means that the maps $\mu_{\mathsf{A}} : \mathsf{A} \to \mathrm{P}$ and $\mu_{\mathsf{B}} : \mathsf{B} \to \mathrm{P}$ are computable algorithms that return a unique property per element, whereas the map $\eta_{\mathsf{AB}} : \mathsf{A} \to \mathsf{B}$ need not be a conversion algorithm. It suffices for the question $(\mathsf{m}_{\mathsf{A}}, \mathsf{m}_{\mathsf{B}}) \in \eta_{\mathsf{AB}}$ to be decidable.

Finally, computing the properties of every pair of elements at the two ends of an interoperability mapping can be delegated to either system, simply because (by definition) the results must match.

- Practically, this means that interoperability allows every computation that depends on the invariant properties alone to be outsourced to the receiving system, with the integrity of the computations being guaranteed up to algorithmic interchangeability. This is fundamental to query-based CAD-CAE, CAD-CAM, and presumably CAD-CAx applications in which computing geometric properties are delegated to the CAD system, while CAx system is concerned with the x-tasks that it is specialized for.[11]

- Theoretically, this is compactly expressed via $\mu_{\mathsf{A}} = (\mu_{\mathsf{B}} \circ \eta_{\mathsf{AB}})$ or $\mu_{\mathsf{B}} = (\mu_{\mathsf{A}} \circ \eta_{\mathsf{BA}})$—using whichever $\eta-$map whose cost (e.g., of translation or querying) is already paid—the properties in one system can be obtained "for free" from the properties computed by the other system.

Following the coloring metaphor used earlier in Fig. 12, the interoperability map can be viewed as a color-preserving transformation, i.e., one that does not leave the color bands specified via $\mu-$interchangeability, as illustrated in Fig. 13. In other words, two systems are interoperable (in either or both directions) if there are guarantees that going back-and/or-forth between them will preserve the invariant properties.

[11] In software development, this practice is commonly known as the principle of 'separation of concerns' (SoC).

### 2.1.3 Integration

Informally, integration refers to a specific form of composing two interoperable systems together, in which the interchangeable elements on the two ends of the interoperability map are associated.

**Property-Based Integration** (Fig. 14)

Let $\mathsf{A} = \langle \mathsf{M_A}; \mathsf{F_A} \rangle$ and $\mathsf{B} = \langle \mathsf{M_B}; \mathsf{F_B} \rangle$ be partially interchangeable systems with respect to $\mathsf{P} = \langle \mathrm{M_P}; \mathrm{F_P} \rangle$ and interoperable via $\eta_{\mathsf{AB}} : \mathsf{A} \to \mathsf{B}$. $\mathsf{A}$ is 'integrated' into $\mathsf{B}$ to obtain a composite system in which the $\eta-$map related elements are paired together, i.e.,

- Given a pair of representations in $\mathsf{M_A}$ and $\mathsf{M_B}$ (denoted $\mathsf{m_A} \in \mathsf{M_A}$ and $\mathsf{m_B} \in \mathsf{M_B}$) associated via $\mathsf{m_B} = \eta_{\mathsf{AB}}(\mathsf{m_A})$, the two constituent representations are paired together into a composite representation $\mathsf{m_{AB}} := (\mathsf{m_A}, \mathsf{m_B})$ of the composite system.
- Accordingly, if an algorithm $\mathsf{f_A} \in \mathsf{F_A}$ is mapped to an algorithm $\mathsf{f_B} \in \mathsf{F_B}$ they are paired together as a composite algorithm $\mathsf{f_{AB}} := (\mathsf{f_A}, \mathsf{f_B})$.

Thus integration is a subset of the Cartesian product $(\mathsf{A} \times \mathsf{B})$ to pairs of elements that not only posses the same properties but also sit at the two ends of the interoperability map.

Once again, in terms of the coloring metaphor used in Figs. 12 and 13, the integration of two interoperable systems amounts to "fusing" them together into a new system in which interchangeable elements in the sending system are migrated to join those of the receiving system (Fig. 14).

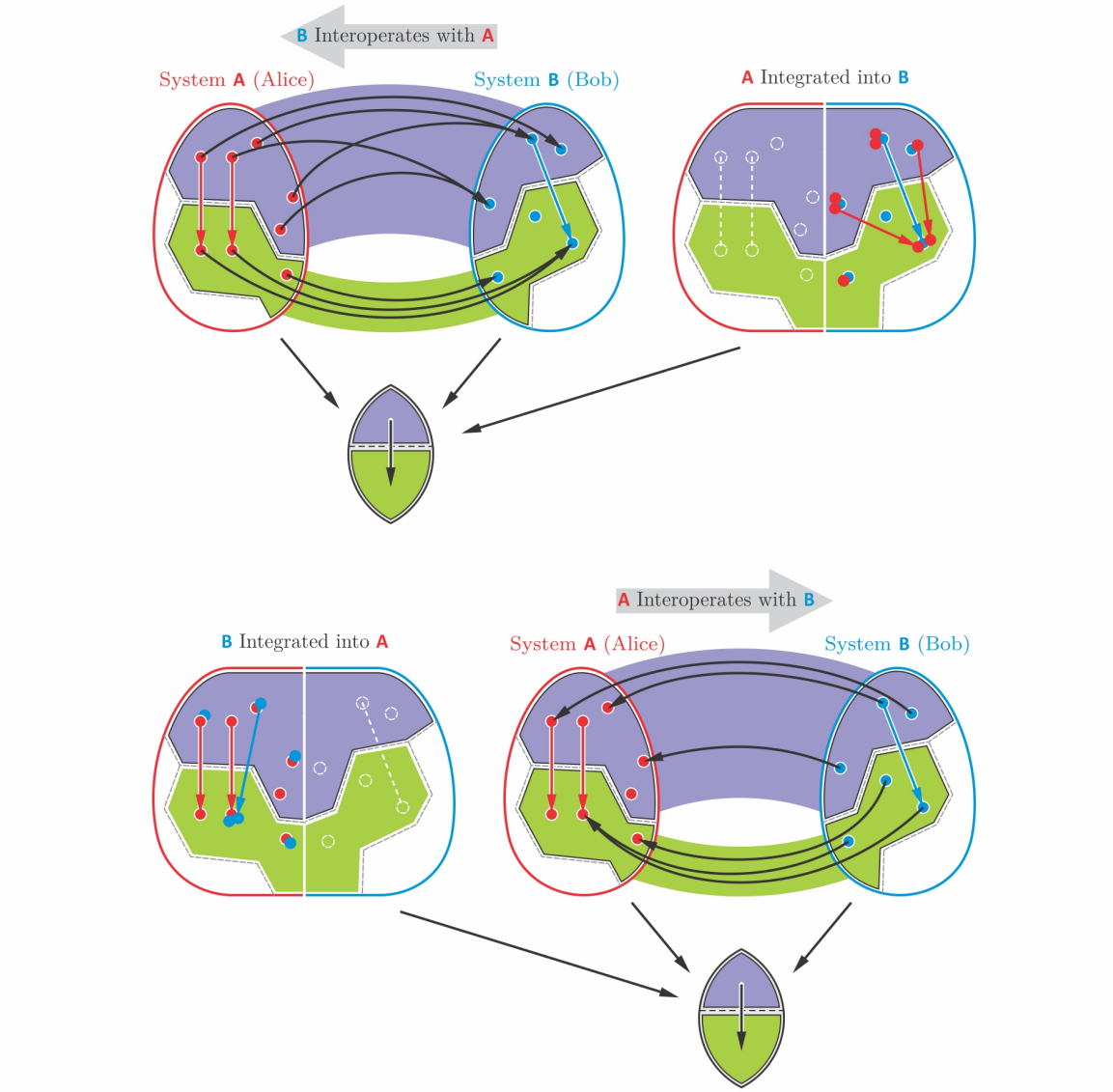



Figure 14: Integration as a pairing of interchangeable elements within the color bands in Fig. 13.

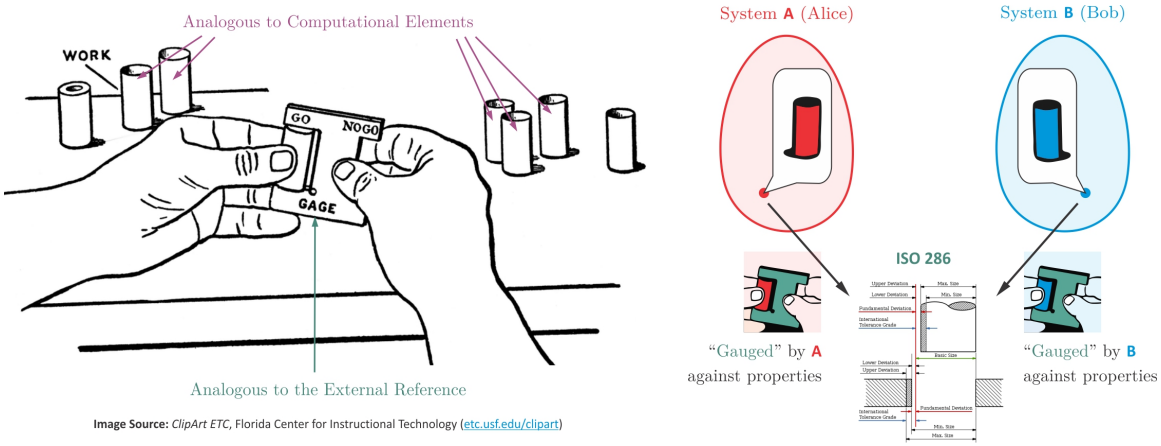



Figure 15: Property-based interchangeability viewed as a (virtualization of) inspection gauging.

## 2.2 More on Invariants★

The philosophical motivation for the Principle of Interchangeability—presented at the beginning of Section 2—can be traced to *part interchangeability* for mechanical assemblies, which was the cornerstone of mass production and economies of scale after the industrial revolution [31]. Although no two parts in a production line can be manufactured to exact congruence, they are deemed interchangeable if one can be replaced with the other without a compromise of form, fit, and function [30]. Historically, parts were certified by using inspection (e.g., 'go/no-go') gauges. Later, the mathematical basis for part interchangeability came into existence with the development and standardization of geometric dimensioning and tolerancing (GD&T) [3]. This basis allows a precise specification of nominal shape and tolerance zones. Any part whose net manufactured shape is within the specified tolerance zones is fit to function and is interchangeable with any other part of such shape, assuming a correct nominal shape design.

To draw an analogy with property-based interchangeability of software systems, if parts are substituted with computational elements (i.e., representations and algorithms), the gauges are virtualized by properties—or more precisely, by the certification of the elements against a set of common properties, as illustrated in Fig. 15. A few lessons can be learned from this analogy:

1. For mechanical assemblies, "function" is assumed to be fully determined by proper "fit" (e.g., standardized into various classes of fit specifications [4]), which, in turn, is accomplished by constraining the "form" (i.e., GD&T specifications). In a sense, form properties serve as *surrogates* to the behavioral properties. By analogy, geometric interchangeability can be used for indirect inference of physical invariants in many CAx-CAx interoperability scenarios.

2. For inspections, parts are *never compared to each other*. This is because for a pair of parts, "being within some tolerance of one another" is *non-transitive*, hence would not yield an equivalence relation.[12] The requirement of transitivity is the historical incentive behind "golden" master parts and gauges. By analogy, it is the primary reason behind the indispensable need for an external reference to specify common semantics for the properties.

[12] An equivalence relation $\equiv$ is one that is reflexive (i.e., $P \equiv P$), symmetric (i.e., $P_1 \equiv P_2$ iff $P_2 \equiv P_1$), and transitive (i.e., $P_1 \equiv P_2$ and $P_2 \equiv P_3$ implies $P_1 \equiv P_3$).

Being within some $\epsilon-$tolerance (e.g., defining $P_1 \approx P_2$ as $\mathsf{d}(P_1, P_2) < \epsilon$) does not yield a transitive relation because $\mathsf{d}(P_1, P_2) < \epsilon$ and $\mathsf{d}(P_2, P_3) < \epsilon$ do not imply $\mathsf{d}(P_1, P_3) < \epsilon$—in fact, the best one can hope for is $\mathsf{d}(P_1, P_2) < 2\epsilon$ due to triangle inequality (assuming $\mathsf{d}$ is a metric) leading to an error accumulation that grows linearly with the number of comparisons. On the other hand, by fixing a "golden" master part $P_0$ and defining $P_1 \equiv P_2$ as $\mathsf{d}(P_0, P_1) < \epsilon$ and $\mathsf{d}(P_0, P_2) < \epsilon$ (at the same time) yields a transitive (thus equivalence) relation with no error accumulation.

3. Parts can be deemed interchangeable with respect to some "features" while being dramatically different (or even specified *incompletely*) with respect to others. Analogously, computational systems can have partial interchangeability with respect to some elements, while others could be different, unspecified, partially specified, or under ongoing development.

Below, we elaborate on the implications of some of the above observations for interchangeability of computational design and manufacturing information.

### 2.2.1 Geometry as a 'Surrogate'★

In CAD-CAx interoperability scenarios, interchangeability is usually specified in terms of shape (i.e., combinatorial, topological, and geometric) invariants alone, since the shape properties is all CAD system can understand and compute. However, in most cases a specific CAx application on the other end must deal with "x-aware" properties that extend well beyond shape information, e.g., involving physical conditions for analysis, planning, material properties, and so on. Nevertheless, most industrial workflow tend to use shape properties as 'surrogate' properties, whose invariance directly or indirectly implies that of 'de facto' properties. This is not surprising because traditionally, shape information serves as a backbone to all product development activities; it is where the design process begins, while physical solutions are not known a priori. For example,

- CAD-CAE interoperability: what we truly care about is interchangeability of the solutions of the initial/boundary value problem—e.g., between the computed FEA simulation on mesh approximation and the hypothetical solution on the exact CAD model. However, since the latter is not available, we assume that interchangeable *specifications* automatically ensure interchangeable *solutions*. For example, in addition to any quality measures imposed by the analysis application, meshing algorithms must preserve the relevant shape properties that support application of boundary conditions and integration algorithms.[13]

- CAD-CAM interoperability: what we truly care about is interchangeability of the solutions of the manufacture process planning problem—e.g., between the 3D printing plan for sliced approximation and the hypothetical planning on the exact CAD model. However, since the latter is not available, we assume that interchangeable *specifications* automatically ensure interchangeable *solutions*. For example, in addition to any quality measures imposed by the planning application, slicing algorithms must preserve the relevant shape properties that are consequential in the spatial reasoning and path planning algorithms.

- Multi-scale material modeling: what we care to preserve when transitioning across the scales is physical behavior (e.g., captured by constitutive relations). However, when physical behavior is not known a priori, it is common to use geometric surrogates from which most physical properties are inferable. In essence, the preservation of the physical behavior across the scales—e.g., between microstructure simulation and equivalent bulk (i.e., average) behavior, related by 'homogenization'—can be often inferred from geometric signatures such as correlation functions, Minkowski functionals, and other deterministic or probabilistic material structure descriptors under certain assumptions (e.g., periodicity and ergodicity).

- CAE and system model interoperability: once again, the interchangeability of the solutions is the ultimate goal; for example, of the partial differential equations (PDE) specified and solved in 4D space-time by the CAE system on the one hand, and of the differential-algebraic equations (DAEs) specified and solved using lumped-parameter models (e.g., used in Modelica), on the other hand. Since the *solutions* are not known a priori, the interchangeability of *specifications* in terms of B/I conditions and stimuli—e.g., between surface distributions for CAE and lumped system-level coefficients, related by 'reticulation'—are relied upon.

[13] However, this may not be enough to assure quality of the computed approximation, and one appeals to additional geometric notions such as mesh quality as a surrogate for properties of the functional analysis space.

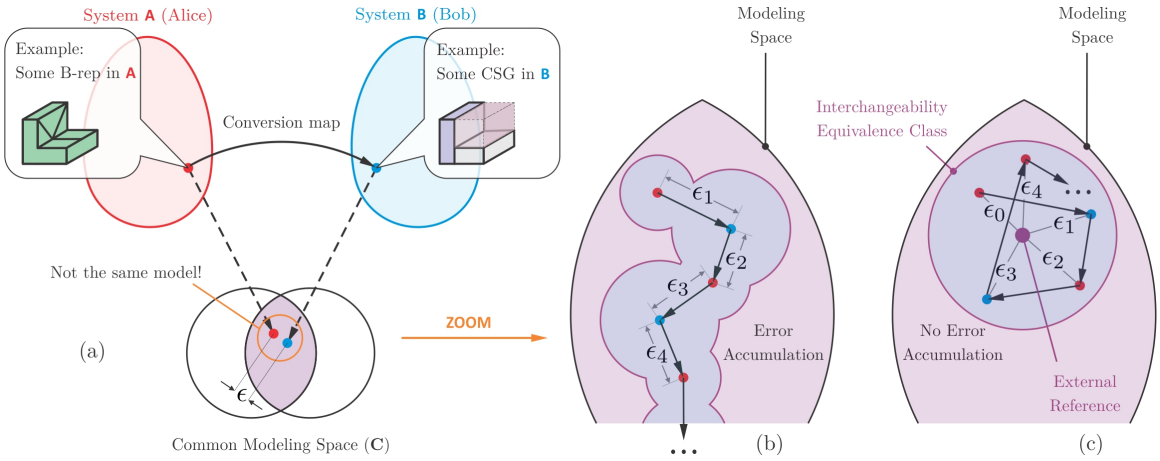



Figure 16: A B-rep to B-rep conversion in (a) with and without error accumulation in (b, c).

We shall elaborate on surrogate properties on a case-by-case basis in Section 4.

### 2.2.2 Transitivity★

The first observation mentioned above reveals a fundamental reason behind the failure of data-centric approaches to interoperability. When two computational systems (e.g., two CAD kernels) use different representations and algorithms, discrepancies are inevitable in their modeling semantics, computing precision, and other factors. Analogous to the incongruent manufactured parts mentioned earlier (Fig. 15 (a)), discrepant computational entities must be compared to a common reference (e.g., a "golden" master) and never to each other. This is not the common practice in most data exchange scenarios. For example, CAD-CAD data exchange facilitated via specialized system-to-system translators or third-party healing/repair utilities come either with implicit assumptions and no guarantees, or at best with guarantees in terms of a two-way comparison. For instance, they could guarantee upper bounds on deviations of the receiving system's converted model (e.g., a rectilinear mesh) compared to the sending system's original model (e.g., a curvilinear B-rep). Similar practices are abundant, in polygonization and mesh generation for CAD-CAE integration or slicing and tool-path generation for CAD-CAM integration, to name a few, most of which are based on a two-way comparison, without appealing to a third-party external reference.

The lack of transitivity starts becoming a problem as soon as computational design tasks are composed into a closed loop for automation. For example, a typical design automation workflow involves a synthesis+analysis loop (e.g., the one in Fig. 1) in which the geometry, topology, and material structure are optimized over hundreds or thousands of iterations, requiring as many data exchanges between CAD, CAE, CAM, material modeler, topology optimizer, and other modules. Every data exchange that merely guarantees upper bounds on the errors/deviations per exchange will suffer from error accumulations that grow rapidly with the number of exchanges.

**Example 2.6.** Consider a repeated geometric data exchange between two systems A and B which are based on different representation schemes (Fig. 16 (a)) and rely on geometric approximation to achieve interoperability. For example, A could model solids bounded by trimmed NURBS surfaces, while B could be restricted to voxelized models or polygonal mesh boundary representations.

The cumulative error can grow without restraint if the guarantees are given by bounding the conversion errors (i.e., $\epsilon_1, \epsilon_2, \cdots$) before and after each conversion (Fig. 16 (b)). And this scenario does not even account for dramatic mutations that can happen at each cycle due to major semantic differences between such systems. On the other hand, if every conversion's accuracy is certified with

respect to a fixed set of externally accessible properties (Fig. 16 (c)) the error can be contained. Such properties can be, but are not limited to GD&T specifications with respect to external datums (e.g., reference points, axes, or planes) rather than internal features, Hausdorff distance from a fixed voxel map, radial basis, or polyhedral approximation, etc.

One could use a third (possibly simpler) system or representation device (e.g., STL format) to specify the reference geometries. It need not be as expressive as A and B, as long as it adequately serves the purpose of providing externally accessible references to measure deviations from.

The utility of using external references to obtain a transitive interchangeability and control error accumulation will be demonstrated with more examples in Section 3.

## 2.3 Classification of Interoperability Problems

Having defined the bolts and nuts of an interoperability scenario (simplified in Fig. 17), we are now in a position to classify the types of scenarios that one deals with into a few major classes. Assuming that we know what the computational elements (i.e., representations and algorithms) in each system are made of, there are two main questions left to answer:

- Do we know the properties that we care to preserve? What are the 'invariants' of the problem?
- Do we have a map (interoperable or not) to related the elements from one system to another?

Depending on the answers to these two questions, four different types of problems typically arise:

1. Verification problem arises when the answer is positive to both questions. The invariants are known, and a map is given—without necessarily knowing how it was developed and what it guarantees. The problem is to verify if the two are consistent; i.e., does the interoperability map preserve the properties?
    - Under restricted conditions or simplifying assumptions, one may be able to theoretically prove the invariance (a priori verification).
    - When theoretical guarantees are not known, one can test the invariance via computational experiments; for example, exhaustive testing on a large enough number of samples or targeted testing for fewer hard benchmark problems (a posteriori verification).
2. Correspondence problem assumes that the target invariants are given, and the question is how to construct a map (either explicitly or implicitly) that establishes correspondence between the models and/or algorithms in the two systems that preserves those invariants.
3. Characterization problem arises often in practice: a correspondence map is already given (perhaps by a helpful translation software vendor or collaborator), but the question is what invariants are preserved by this map? Without further qualifications, the question may appear academic, but at the very least should be able to characterize the correspondence map in terms of the properties that are critical to the application in hand.
4. Innovation: is when nothing is given, as with emerging applications and innovative solutions. The goal is to complete the picture iteratively and convert it to one of the above problems.

The idea is illustrated in Fig. 18. The problems grow harder as we move down the list. Eventually, the goal is to complete the picture and convert the problem to one of verification, which is more straightforward—at least conceptually, but not necessarily computationally.

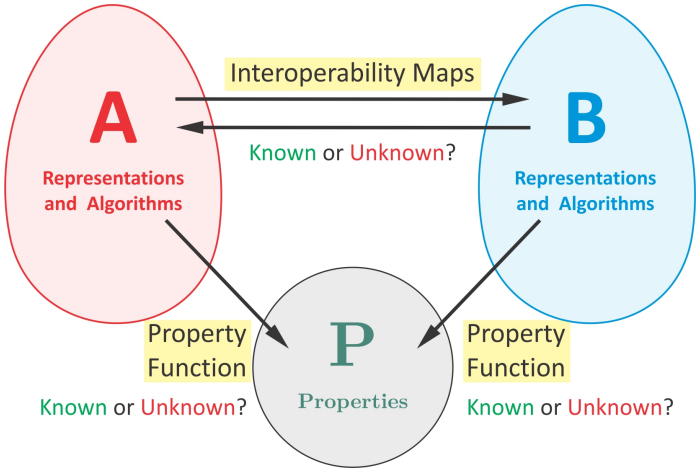



Figure 17: The simplified anatomy of an interoperability scenario: what parts do we (not) know?

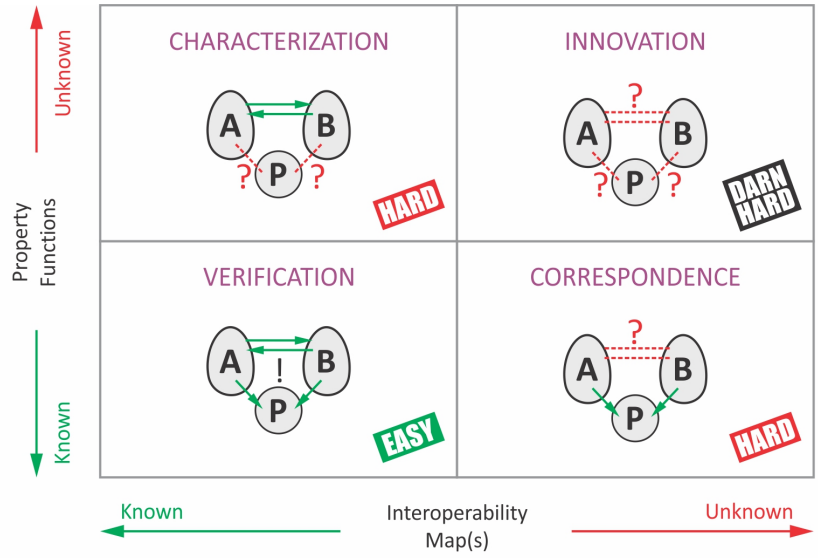



Figure 18: There are four major types of interoperability scenarios depending the known/unknown status of the invariant properties and interoperability map (shown on the two axes).

# 3 Use-Case Scenarios: CAD-CAD Interoperability

> "Therefore geometry is founded in mechanical practice, and is nothing but that part of universal mechanics which accurately proposes and demonstrates the art of measuring."
>
> Isaac Newton, Principia, 1687

In this section and the next, we present a few concrete interoperability scenarios along with their advantages, challenges, and best practices.

We begin with the more familiar case of CAD-CAD data exchange, not only because it is the most studied and the best understood, thanks to decades' worth of research efforts, but also in recognition of the importance of geometric properties to surrogate non-geometric properties. This role has lead to the historical role of geometric modeling as a medium to support various product development activities ranging from conceptual design and detailed modeling to shape synthesis, analysis, process planning, optimization, inspection, documentation, and archival.

The interoperability challenges and scenarios discussed in this section readily apply to data sharing and exchange between solid modeling software, ranging from commercial packages to open-source libraries or in-house solutions, shape synthesis tools (e.g., for shape and topology optimization), data acquisition tools (e.g., point clouds obtained from 3D cameras, scanners, or CMMs), and other geometric representation media.

Once the range of possibilities and challenges for geometric interoperability are understood, we will discuss CAD-CAx interoperability in Section 4.

**Synopsis: Use-Case Scenarios**

To review the terminology presented in Section 2 and provide an outline, we will discuss the interoperability scenarios for two systems, 'Alice' $\mathsf{A} = \langle \mathsf{M}_\mathsf{A}; \mathsf{F}_\mathsf{A} \rangle$ and 'Bob' $\mathsf{B} = \langle \mathsf{M}_\mathsf{B}; \mathsf{F}_\mathsf{B} \rangle$, color-coded as such in all notations and figures:

- Each system has its own set of representations ($\mathsf{M}_\mathsf{A}$ and $\mathsf{M}_\mathsf{B}$) and algorithms ($\mathsf{F}_\mathsf{A}$ and $\mathsf{F}_\mathsf{B}$) which may or may not implement a common modeling space—i.e., interpret to the same or different models and/or functions, respectively.
- The property-based interchangeability is tested by means of a pair of property functions $\mu_\mathsf{A} : \mathsf{A} \to \mathsf{P}$ and $\mu_\mathsf{B} : \mathsf{B} \to \mathsf{P}$, whose semantics are specified with respect to a common property space $\mathsf{P}$. Each system is separately responsible for correct interpretation and implementation of its property function by adhering to the common specification.
- Each interoperability scenario is characterized by the properties in $\mathsf{P}$ that (supposedly) remain invariant under the action of an interoperability map $\eta_{\mathsf{AB}} : \mathsf{A} \to \mathsf{B}$, which is a functional correspondence between interchangeable elements (from $\mathsf{A}$ to $\mathsf{B}$).

Accordingly, characterization (i.e., finding invariants), correspondence (i.e., establishing a map), and verification (i.e., testing if the map preserves invariants, as it is supposed to), are discussed.

## 3.1 CAD-CAD Data Exchange Properties

The discussion in this section does not focus on any particular properties. As we discussed in the last section, these properties are and should be determined by specific interoperability needs and applications. However, for illustration purposes we may invoke examples from the following broad classes of "shape" properties:

- *combinatorial* properties: the topological structure of the underlying cell complex or parts of it that are precisely preserved (e.g., simplicial structure, incidence/adjacency relations, etc.);
- *topological* properties: the topological structure of the shape itself, even if it is described using a different cell decomposition (e.g., homology, homotopy, homeomorphism, manifoldness, etc.);
- *geometric* properties: the metric properties (e.g., distances, sizes, and tolerances), affine properties (e.g., colinearity, parallelism, and convexity), group properties (e.g., symmetry), etc.;
- *differential* properties: the local neighborhood properties of the shape and/or its boundary (e.g., normal and curvature constraints, $G^k-$continuity, etc.);
- *integral* properties: measures (e.g., cardinaltiy, curve length, surface area, and solid volume), moments, Euler characteristics, Minkowski functionals, etc.;

and possibly others, depending on specific applications. For example, what is typically referred to as 'design intent' is often encoded into parametric CAD models as sketch or feature constraints (e.g., cocentricity, coaxiality, parallelism, perpendicularity, tangency, etc.) or dimensioning and tolerancing (e.g., size, angle, and freeform) can be characterized as combinations of the above classes of properties. The classification itself (e.g., whether a property is topological or geometric, or both) is not of primary concern. The main point is that the properties that one wishes to preserve must be explicitly specified. Some properties are qualitative, enforced via standard equivalence relations (e.g., homeomorphism) without the need for redefining the property function from scratch. Some will depend on quantitative, but universally well-defined property types (e.g., scalar-, vector-, or generally tensor-valued) and property functions (e.g., measures or integrals). Others require explicit specification of a common reference, recognized and accessible by both systems, with respect to which certain measurements need to be made to enforce constraints (e.g., upper bounds on Hausdorff distance or GD&T deviations). One way or another, property-based interchangeability has to satisfy the *transitivity* requirement discussed in Section 2.2.2 and exemplified below for geometric cases.

## 3.2 Transitivity via External References

Importantly, we also recognize that most of geometric data is inherently imprecise, and numerical algorithms operating on it involve inevitable approximations.[14] This means that the properties, with respect to which interchangeability and interoperability are certified, should accommodate tolerances to data errors/noise and other inaccuracies. As discussed in Section 2.2.2, it is crucial that all approximation and tolerancing schemes rely on *external references* in order to guarantee an equivalence relation (by enforcing its transitivity), or at least to control error accumulations and model degradation in iterative data exchange scenarios. For example, tolerances specified with respect to an internal feature (e.g., some B-spline edge or face) in a NURBS-based B-rep system A cannot be properly interpreted by a CSG or polyhedral B-rep system B. Even if the two systems appear to have similar representational capabilities (e.g., both understand B-splines), there are a number of unsolved issues with persistently addressing internal references (e.g., persistent naming) that make external references a safer choice with respect to which deviations can be quantified and constrained in various manners. Example quantifications are upper bounds on Hausdorff distance

[14] Arguably, this is the primary reason why interoperability has remained an unsolved problem over the decades.

to common mesh approximations, GD&T tolerance specifications using common datum references, constraints on integral properties averaged on voxels over a fixed common grid, membership and/or distance specifications over a densely sampled common point cloud (with or without tolerances), and so forth.

In general, defining interchangeability with respect to qualitative (e.g., combinatorial) properties are less subjective than quantitative (e.g., metric) properties, because the former typically do not have to deal with precision, accuracy, and tolerances. Examples of the former are logical expressions and graph-like structures that define incidence/adjacency relations in a mesh. The need for additional references such as GD&T datums, voxel maps, point clouds, and other neutral geometric constructs are mostly exclusive to quantitative (e.g., metric) properties of the shape.

### 3.3 Use-Case Schema #1: CAD-CAD Data Exchange

Here we present the six most common approaches to geometric interoperability—tagged Use-Cases #1.1–1.6. While it is conceivable that other scenarios can be generated, either obtained as combinations of these methods or constructed from scratch using the general principles given earlier, the following is an assortment of representative examples than spans a broad range of possibilities:

1. **Use-Case #1.1:** standardizes on the system, using the same representations and algorithms. It leads to trivialized conditions of interoperability in which virtually all properties are preserved (i.e., model-based interoperability), but is limited in its applicability, expendability, scalability, and ability to support innovation.

2. **Use-Case #1.2:** standardizes on the representations, using possibly different algorithms. It leads to broader and straightforward (even if nontrivial) interoperability that preserves properties computed using the common subset of algorithms, but remains fairly limited. Arguably, without algorithm interchangeability, same representations are not really the same due to incomplete semantics coming from algorithmic inconsistencies (i.e., how the data is being used).

3. **Use-Case #1.3:** uses system-to-system translators that are optimized to preserve as many properties as possible, based on a deep (e.g., developers') knowledge of their internal architectures. It leads to the most widely used industrial solutions, but lacks formal guarantees, sees unpredictable failure rates, and requires recreating translators for every pair of systems.

4. **Use-Case #1.4:** standardizes on neutral formats, and uses system-to-/from-neutral file translators. It simplifies the process by requiring every new system to interoperate with the standard format, but at the same time limits the preserved properties to those inferable from the neutral format, leading to larger failure rates in the presence of conversion errors and inaccuracies.

5. **Use-Case #1.5:** standardizes on generic recipes (i.e., procedural or declarative "programs"), and uses a common language to exchange largely symbolic construction recipes. It minimizes numerical data content (hence susceptibility to numerical errors) of the exchange mechanism itself, at the expense of overlooking the possibility of inconsistent interpretations of each system when internally instantiating/evaluating the recipes.

6. **Use-Case #1.6:** standardizes on a collection of queries (i.e., functional semantics), usually in a form of formally defined APIs. It eliminates error-prone translation altogether and also prevents semantic misinterpretations. However, the preserved properties are restricted to those computable from a finite composition of queries. Its power, formal properties, and practical limitations are yet to be understood and validated.

In each use-case, we examine the landscape and provide concrete examples, carefully analyzed in terms of which properties can be preserved, what the property functions look like, what the interoperability map is, and how it can be verified. The aforementioned attributes, advantages, and drawbacks, are put into perspective followed by guidelines (i.e., "best practices").

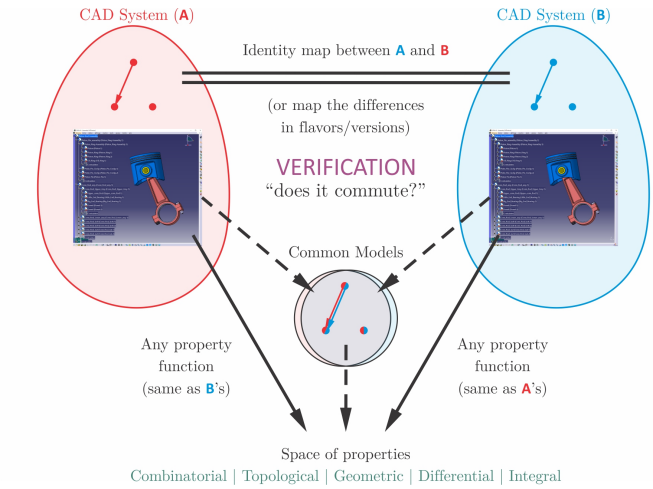



Figure 19: Use-Case #1.1: When the same representations (bullets) and algorithms (arrows) are used in both ends, the verification of invariance becomes trivial for both: all properties are preserved.

### 3.3.1 Use-Case #1.1: Single System Solution (Same Representations & Algorithms)

**Basic Idea/Setup.** Using the *same* CAD system is the safest and simplest interoperability solution (i.e., $\mathsf{A} = \mathsf{B}$), including both representations (i.e., $\mathsf{M}_\mathsf{A} = \mathsf{M}_\mathsf{B}$) and algorithms (i.e., $\mathsf{F}_\mathsf{A} = \mathsf{F}_\mathsf{B}$). This scenario should not be confused with standardizing on the same representation scheme (e.g., NURBS-based B-reps) or algorithmic methodology (e.g., isogeometric analysis).[15] Here, the idea is to use the same CAD software/kernel to power all activities in a workflow.

A slightly different scenario is obtained when $\mathsf{A}$ and $\mathsf{B}$ are different "flavors" or "versions" of the same system, due to small (and carefully documented) nuances between their representations (e.g., $\mathsf{M}_\mathsf{A} \neq \mathsf{M}_\mathsf{B}$) and/or algorithms (e.g., $\mathsf{F}_\mathsf{A} \neq \mathsf{F}_\mathsf{B}$). As long as they have the same core data structures and algorithms, the following discussion applies *only* to the core subsystem.

**Examples.** Standardizing on the system is a simple and viable (but limited) solution to both vertical and lateral interoperability (Fig. 1):

- For vertical interoperability (same workflow, different tasks), it is common for companies, universities, or research labs to choose a particular PLM software package, or standardize on a geometric kernel, for all computational tasks throughout the entire enterprise.
- For lateral interoperability (different workflows, same tasks), common choice of systems is convenient as it minimizes the need for additional manual adjustments and/or collaborative software development to make sense of exchanged data between two enterprises. All it takes is to send a file—if it works using $\mathsf{X}$ in one end, it must work using $\mathsf{X}$ in the other end as well.[16]

In fact, the viability of the PLM business model largely relies on its safe solution to (in-house) interoperability, in spite of its clear limitations in scalability to heterogeneous systems and services.

[15] In other words, a statement such as "use NURBS-based B-reps for all tasks" is not strong enough for this scenario.

[16] For examples, OEM suppliers (e.g., for automotive companies) may have no choice but to purchase the same CAD system as their customers to be able to interoperate with them.

**Properties.** Any property that is computable from the information content of the common representations in $\mathsf{M}_\mathsf{A} = \mathsf{M}_\mathsf{B}$ is automatically preserved across all components using the same system/kernel, noting that algorithms in $\mathsf{F}_\mathsf{A} = \mathsf{F}_\mathsf{B}$ that compute those properties are also the same. In other words, using the same system in fact achieves the model-based interoperability (discussed in Section 2.1.1) for all valid representations and algorithms within the system.

Accordingly, the property functions include every combination of computable functions that the system offers, as long as the same ones are chosen for both (i.e., $\mu_\mathsf{A} = \mu_\mathsf{B}$). For example, if both systems are based on a CAD kernel X, every program that calls the same sequence of X subroutines—and noting else except trivially equivalent instructions and assignments—must return the same results for the same data in both systems. This in effect is a trivial solution to the characterization problem because all properties are invariant.

**Interoperability Map.** Obviously, the interoperability map is the *identity* function on $\mathsf{A} = \mathsf{B}$. This is a trivial solution to the correspondence problem. For example, if both systems use the same exact CAD kernel X, this simply means that the data is passed identically from one system to another without any changes or conversions. Since algorithms are also identical, the semantics remain intact, too.

**Verification.** The verification problem is also solved trivially, since the identity function preserves all properties. This is the simplest example of when a theoretical proof exists, eliminating the need for sampling or benchmarks, as discussed in Section 2.3.

Note, however, that this excludes a different type of verification pertaining to the proper *usage* of the same system—e.g., checking if both systems that are based on X are actually using X correctly and consistently. In other words, we are not talking about internal integrity of the systems, which is largely taken for granted for widely tested commercial systems, but to a lesser extent for open-source libraries, beta-prototypes, and work-in-progress.

**Advantages.** The main advantage of this approach is the finest form of (i.e., model-based) interoperability. The key is an assumption that the system internally guarantees a sound implementation—in the sense that equivalence classes of representations and algorithms within the system are well-defined and properly used.[17] This is a reasonable assumption for commercial-strength systems and seasoned users, and is a matter of practicality; but it should not be taken for granted, as a number of fundamental problems remain unsolved (e.g., robustness of geometric operations).

**Drawbacks.** The approach essentially avoids most of the challenging interoperability issues, but is limited in scope and applicability as it does not support many common and important scenarios:

- Standardizing on an existing technology and system is at odds with innovation, which often depends on new representations and algorithms invented and optimized specifically to support new approaches and new applications.
- There is no guarantee (or even reason to believe) that solutions based on adaption of a single system will interoperate with any other system.[18]
- The scope of interoperability is limited only to a common core (representations and algorithms)—in both vertical and lateral interoperability scenarios. Thus, broad claims of interoperability do not apply to different versions of the same system or different tasks relying on different assumptions and/or algorithms.

[17] Note that there are usually multiple ways to perform the same computations in the same system. Different users or application may use the same system differently, and the system must guarantee internal interchangeability.

[18] Indeed, such interoperability prospects are often intentionally restricted as a matter of business strategy.

In summary, this approach simplifies interoperability in the short-run and within a restricted scope, but does so at the expense of scalability and innovation, in the long-run. In particular, it does not leave any "hooks" for future interoperability with any other system or enterprise solutions.

**Best Practices.** To qualify under this use-case scenario, one should use the exact same representations and algorithms, avoiding different flavors as much as possible. If different flavors or versions of the same system are used, one must be very clear about the core components that are common between them as well as the changes—not just in syntax but also in semantics. The properties must depend solely on the shared core, if they are to enjoy the trivial a priori verification. Invariance of all other properties may require a posteriori verification.

A common pitfall is the temptation to assume that similar flavors of (presumably) the same representations and algorithms interoperate, without going through an explicit enumeration of their assumptions and constraints. This fails because the interpretation inconsistencies can and will compromise the invariance of properties.

One must be cautious about improvements (e.g., version control) and never assume they will not affect future use in downstream applications. Backward compatibility should be documented in the language of preserved properties and the mapping from each version to the next.

---

### 3.3.2 Use-Case #1.2: Standardizing on Representations (with Different Algorithms)

---

**Basic Idea/Setup.** The next use-case is for $\mathsf{A} \neq \mathsf{B}$, when both systems still use the exact same representations (i.e., common internal data structures) as before (i.e., $\mathsf{M}_\mathsf{A} = \mathsf{M}_\mathsf{B}$). But they may use them inconsistently via different algorithms (i.e., $\mathsf{F}_\mathsf{A} \neq \mathsf{F}_\mathsf{B}$). This could be due to different algorithms implementing (presumably) the same function, which may come with different guarantees (e.g., different accuracy). This could also be due to new operations being added, old operations going obsolete, or improvements/modifications of previous algorithms in evolving implementations.

**Examples.** A frequently encountered example is of two systems that are capable of working with the same representation scheme (e.g., NURBS-based B-reps, polygonal mesh B-reps, voxel maps, etc.) or data formats (e.g., STL), but are not necessarily equipped with the same algorithms. Some algorithms may not exist in either system—e.g., A is equipped with topological operators (e.g., Euler operators) while B can only make geometric modifications (e.g., moving vertices or control points). In these cases, it is clear that such algorithms are not included in interchangeability classes.

A more subtle situation where the answer is not as clear is when both systems are equipped with the same geometric operations (e.g., testing membership, computing bulk properties, predicting intersections, etc.) performed in completely different ways. For instance, the same point membership classification operation on B-reps can be implemented using ray tracing, winding numbers, or a number of other techniques. These algorithms, despite approximating the same mathematical function, may come with completely different behaviors and guarantees in terms of accuracy and applicability.

**Properties.** Any property that is computable from the information content of the common representations in $\mathsf{M}_\mathsf{A} = \mathsf{M}_\mathsf{B}$ is preserved, only if at least a *subset* of algorithms in $\mathsf{F}_\mathsf{A} \neq \mathsf{F}_\mathsf{B}$ that are sufficient to compute those properties are also chosen to be the same. This special subset of common algorithms are specifically charged to compute the property functions, and their identical implementation in both systems guarantees property-based interchangeability of representations on which they compute. If this subset is extended to the entire algorithm space (i.e., $\mathsf{F}_\mathsf{A} = \mathsf{F}_\mathsf{B}$), the interoperability scenario transforms to that of Use-Case #1.1.

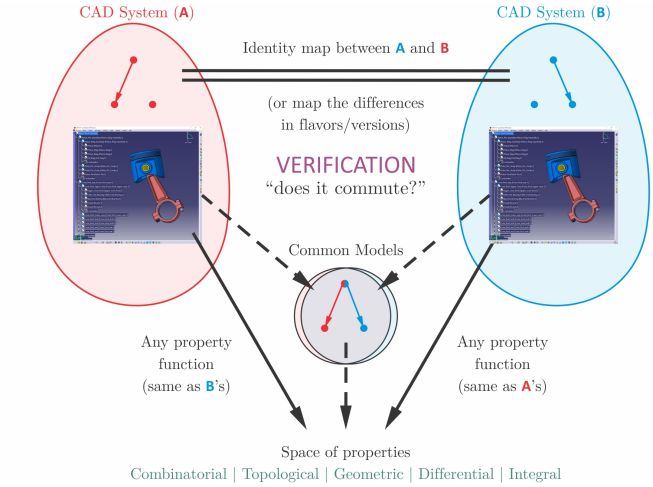



Figure 20: Use-Case #1.2: When the same representations (bullets) are used in both ends, with possibly different algorithms (arrows), the verification of invariance is no longer trivial for the latter: all properties are preserved as long as they are computed using the same algorithms.

For example, it is entirely possible that two systems using identical representations or data formats could compute inconsistent results on them, be it point membership test, integral properties, metric or topological properties. But if the membership algorithms are identical, other computations that depend solely on that membership computation can be made consistent as well. If all other computations are purposefully composed from membership tests alone, algorithm interchangeability is achieved.[19] But this is rarely the case in practice, resulting in partial algorithm interchangeability. In essence, the characterization problem amounts to identifying the proper subset of identical algorithms that cover the properties that we care about.

**Interoperability Map.** The interoperability map for the identical representations is the identity function (similar to Use-Case #1.1). The limitation is that interchangeability and interoperability are restricted to the representations in $\mathsf{M}_A = \mathsf{M}_B$ but not necessarily the algorithms in $\mathsf{F}_A \neq \mathsf{F}_B$—except, of course, the identical subset of algorithms that implement property functions. If the common subset is known, the solution to the correspondence problem is straightforward for representations, but limited in applicability to the remaining (i.e., non-identical) algorithms.

For example, if point membership test is used to define the properties, computing the test on the same representations using the same algorithm is automatically consistent. But the guarantees that one can obtain on a different computation (e.g., distance to boundary) are not trivial unless the distance algorithm does not make calls to anything other than the said membership test.

**Verification.** The verification problem is also solved trivially or in a straightforward fashion, when there is a priori guarantee of invariance of properties that depend on the common representations alone—or the common core component in different flavors or versions. This excludes algorithms that are different in the two systems, for which a posteriori verification may be required—i.e., certifying if the two different implementations of the same function are interchangeable, by checking if they preserve the relationship between input/output properties.

[19] With this special composition, the interoperability scenario transforms to that of Use-Case #1.6, where the burden of model-based interchangeability is shifted from representations (of supposedly identical models) to algorithms (of supposedly identical functions), to which we shall return.

For example, if the documentation accompanying a version change of a CAD kernel X shows that all geometric constraints for sketching (e.g., symmetry, parallelism, coaxiality, etc.) are implemented in the same fashion as before, the user can trust that every property of a model that relies on those constraints will remain invariant. However, if the update had to do with the way constraints are enforced or solved, it may be necessary to test the invariance of the properties one-by-one—e.g., on samples that sufficiently spans the space of models and/or constraints.

**Advantages.** Standardizing on representations (or core data structures) is the next safest and easiest approach after Use-Case #1.1 when one needs algorithmic variety. Consequently, it supports *some* innovation, at least more than the previous use-case, within the limitations of the preserved structures. Although reasoning about the interoperability map is not always trivial, it is straightforward for the representations (or common subset of algorithms) when the differences are properly documented, though not so much for the remaining algorithms.

**Drawbacks.** This approach inherits many limitations of the Use-Case #1.1:

- Using the same representation schemes still limits flexibility and innovation to a great extent.
- Although representation interchangeability is obtained for free, there is still no straightforward path to guarantee algorithm interchangeability. This often creates a misleading appearance of interoperability, which in fact may not be adequate.
- One again, there is no guarantee (or even reason) to expect interoperability with any other systems that do not use the same representations.

Discrepancy in the algorithms of the two systems becomes especially problematic in the face of robustness problems. For example, even though different PMC algorithms (e.g., using ray tracing, winding numbers, etc.) on the same representations could yield different results, it is expected that they can be made interchangeable by picking the right tolerances to errors near the boundary. This is not necessarily true for intrinsically non-robust computations [18], for instance, when computing minimum feature size (i.e., distance to medial axis). For such computations, small upstream perturbations can lead to large downstream discrepancies.

**Best Practices.** To qualify under this use-case scenario, the two systems must share the exact same representations. Once again, if different flavors or versions of the same system are used, it is important to carefully document the common core and differences, and look for invariant properties that are known to depend on the common core. For all other properties, their invariance should be checked in a posteriori verification.

A common pitfall is the temptation to assume that representation interchangeability implies algorithm interchangeability. It is possible that different versions of the same system use the same representations in entirely different ways, and despite syntactic similarities, algorithms may not be interchangeable. The latter can be discovered through a posteriori verification.

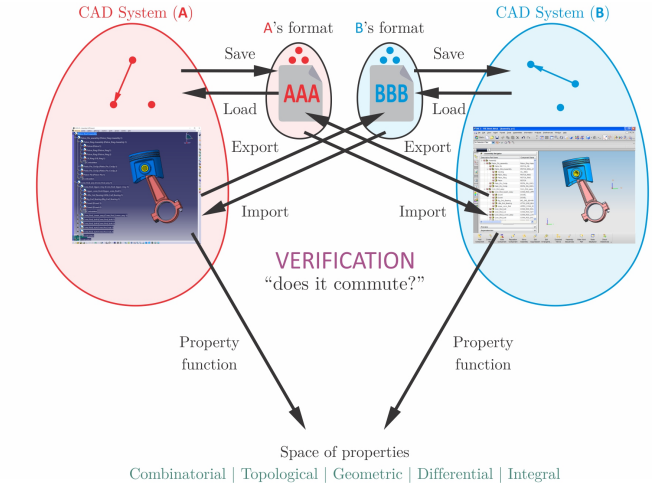



Figure 21: Use-Case #1.3: Representations (bullets) are converted via export/import to/from A and B's file formats, whose interchangeability can be verified by testing them against the properties that should be preserved. Interchangeability of algorithms (arrows) remains a challenge.

### 3.3.3 Use-Case #1.3: Data-Centric Exchange (via System-to-System Translators)

**Basic Idea/Setup.** Direct system-to-system representation translation is one of the most popular interoperability solutions that is widely embraced by industry. The translation software are often designed to take into account not only the specific representation differences, but also a knowledge of how representations are used by the interoperating systems.

Note that translation in this use-case is by direct translators and does not require an intermediate (e.g., neutral) file format (Use-Case #1.4).

**Examples.** Most product lifecycle management (PLM) suppliers often provide export/import support for other systems' file formats. When representation schemes in the two systems are similar (e.g. B-reps with the same type of curves and surfaces), translation amounts to matching the correct data types in the two systems. Even though these translator are improved over time with new versions, failures do occur (e.g., due to incompatible precision semantics). Moreover, as in Use-Case #1.2, interchangeability of the two representations is not sufficient to guarantee interoperability of the systems developed with different assumptions and algorithms.

Hence in practice, "interoperability solution" providers enhance the translation with a variety of data 'quality' diagnostics and validation tools, as well as repair and simplification for model re-use in other CAD systems or downstream CAE/CAM applications. It is often unclear what metrics of quality are used, and what properties are assumed indispensable for model re-use, while others can be forgone. More importantly, the quality checking, comparison, validation, and repair utilities almost never appeal to external references to ensure transitivity (see Section 2.2).

When representations schemes are very different (e.g. B-reps versus voxel maps or CSG trees), translation takes on the form of representation conversion. Representation conversions may or may not be always possible, could be one- or bi-directional, and could be either lossless or lossy. Important example of data-centric lossy translation are polygonization/meshing of curved surfaces and volumes,

voxelization of arbitrary volumes, slicing 3D volumes into stacks of 2D surfaces, hierarchical bounding volume (HBV) approximations such as oriented bounding box (OBB) trees, octrees, and sphere-trees, point cloud sampling, and other shape approximation schemes. Each simplifies the shape and captures some (but not all) of it properties for a particular purpose, e.g., faster collision detection, easier motion planning, discretization for downstream CAE/CAM, etc. Each simplification can be conceptualized in terms of the information that is preserved (i.e., invariant properties) as it is critical for the particular purpose, at the expense of losing information that is disposable.

**Properties.** Unlike the previous use-cases, there is no reason to expect that model properties (or those of a dominant core) will be preserved by the translation process when the systems use dramatically different assumptions, semantics, and representation schemes. Different translators and convertors focus on preserving some properties while inevitably sacrificing others. There has to and will be trade-offs between combinatorial, topological, geometric, differential, and integral properties exemplified at the beginning of this section. In this use-case, the properties need to be determined one-by-one by a careful and exhaustive analysis of the translation algorithm, which amounts to a non-trivial characterization problem.

For example, consider converting between two CAD kernels directly via export/import to/from their native file formats. The developers may intend to preserve the net shape properties, without guaranteeing the same construction history. They may additionally choose to preserve the shape constraints that capture the design intent if both systems support the same semantics. In either case, one should explicitly enumerate the properties that are preserved as well as the ones that will be lost in the translation—e.g., due to lack of support or different semantics in the receiving system. In addition, one should recognize the widely common numerical errors in translation as well as the differences in representational precisions of the two systems. The interchangeability of measurements (e.g., distance, size, angle, etc.) performed on the representations before and after translation can be qualified with respect to external references—e.g., tolerance specifications with respect to established datums, Hausdorff distance to a fixed point cloud. etc. This is where the trade-offs will be decided—e.g., how narrow the tolerance zones can be chosen without compromising the invariance of topological properties.

Furthermore, unlike the previous use-cases, there is no common representation/core to rely on for computing the property functions. The property functions are computed via entirely different algorithms for different representation schemes (e.g., for B-rep versus CSG) and the burden of each function's correctness is on the authoring system. There is no way to prescribe how to implement them in general, and the success of data-centric interoperability depends on how successfully each system delivers its property function.

**Interoperability Map.** The interoperability map in this use-case is a representation conversion, usually implemented as a format translation process, (export+import operations via files). Operationally, it can be done in several different ways, for example:

- A saves to its own file format, and B knows how to import A's format.
- A knows how to export to B's file format, and B loads its own format.
- A third-party system knows how to import A's format and export it as B's format—making it their business to maintain a knowledge of both systems' assumptions and constraints—so that A and B save and load to their own formats without worrying about conversion.

All three options may be available for dominant CAD systems. As a matter of business strategy, it is also common for system A to support native import of models created in system B, without providing any means for exporting models back to B.

**Verification.** A priori verification in this use-case is difficult, if not impossible, as theoretical guarantees for the correctness of conversion algorithms are limited. Even when translation does not involve representation conversions, precise semantics of interoperability is implicit in selection of data types, numerical accuracy of algorithms, assignment of tolerance zones, and other specifics internal to the interoperating systems.

To perform a posteriori verification, on the other hand, one needs to show that the conversion does not compromise the invariance of the properties. For example, when converting between two different CAD kernels, assuming that property functions and translators (i.e., save+import, export+load, or third-party solution) are both available, the verification amounts to testing if the properties (computed separately by each system) match before and after translation. The same can be done for polygonization/meshing, voxelization, reconstruction, etc.

If the interoperability maps are given in both directions $\mathsf{A} \rightarrow \mathsf{B}$ and $\mathsf{A} \leftarrow \mathsf{B}$, converting back-and-forth between the two systems, no matter how many times performed, must keep the model within the same interchangeability class. As a matter of fact, one can precisely quantify model 'degradation' (in one conversion or multiple round trips) in the language of preserved or compromised properties. If the first attempt at verification fails (e.g., due to error accumulation) one can either redo correspondence by revisiting and improving the translation (i.e., interoperability mapping); or redo characterization by relaxing the interchangeability conditions—i.e., revisiting the properties to obtain a more coarse-grained partitioning into interchangeability classes. For instance, for each sampled model, one can perform membership/distance tests over a large number of points (e.g., on a fixed grid) and compare how the representations in each system respond. If verification fails in a larger fraction of tests than desired, one can relax the constraints (e.g., use larger tolerances), and try again.

**Advantages.** Arguably, direct system-to-system translators have offered the most effective and practical solution to date to CAD-CAD data exchange. This is primarily because one can exploit the specific characteristics of the particular sending and receiving systems to maximize the efficacy of interoperability—unlike the case with neutral file formats (Use-Case #1.4). The caveat is that this approach to interoperability does not provide guarantees without characterization, i.e., unless the invariant properties are explicitly worked out and documented by the translator developers.

Compared to the next two use-case scenarios, direct translation allows semantic interoperability by directly appealing to the shape properties than one cares about, rather than relying on lossy intermediate structures (i.e., "semantic bottlenecks") such as neutral file formats (Use-Case #1.4) or procedural recipes (Use-Case #1.5). In other words, the rich information content of the sending and receiving systems' representation schemes (as much as it is supported by both) can be preserved—e.g., shape constraints that capture the design intent, physical annotations (e.g., for CAE), tolerance information (e.g., for CAM), etc. As systems evolve in complexity and expressive power in future versions, the translators need to evolve as well to accommodate mappings between newer features.

**Drawbacks.** Direct system-to-system translation has also a few significant disadvantages, namely:

- To move beyond syntactic interchange of data, direct translation must have intimate knowledge of internal workings of the interoperating systems, and possibly access to internal data structures and conversion algorithms/source-code.
- Even with such knowledge, precisely characterizing what is being preserved is usually difficult or impossible, when the systems' properties and assumptions are not fully documented.
- Trade-offs need to be made when deciding which shape properties (e.g., combinatorial, topological, geometric, differential, and integral) to preserve, which is challenging—especially for CAD-CAD data exchange where particular downstream application is not known at the time of translation

- Once again, representation interchangeability does not guarantee algorithm interchangeability. The receiving system may still use the model for tasks it is not meant for. For example, a smoothed model for visualization may not be precise enough for manufacture planning.
- Last but not least, the number of translators grows quadratically with the number of systems.[20] In addition to being costly and inefficient, this approach to interoperability also stifles innovation, as introduction of any new system demands a large number of translators.

The most common reason for the failure of this scenario is the lack of guarantees or explicit enumeration of invariants. A more subtle issue is that even when there are well-described guarantees for data quality, they are almost always based on a comparison of the 'downstream' model (in B) with the 'original' model (in A), rather than an external 'master' model. As discussed in Section 2.2.2, this compromises transitivity and provides ground for error accumulation in iterative applications with hundreds or thousands of data translation events (see Example 2.2.2).

**Best Practices.** Direct translation interoperability solutions should begin by explicitly enumerating the invariants, then providing the correspondence solution that preserves them. Otherwise, the characterization problem (i.e., finding the preserved properties) of a given translator poses serious unsolved challenges. The specification of invariants also allows rigorous formulation of verification protocols for developing third-party diagnostics and repair tools.

In case of black box translators, without source-code or proper documentation of the invariants, characterization becomes a matter of trial-and-error, which is unfortunately the current norm in industrial applications. There exist rich catalogues on model degradation and CAD data 'quality' using various taxanomies of qualitatively different modes of failure. They range from syntactic vs. semantic, topological vs. geometric, global vs. local, to other classifications including notions of punctures (e.g., holes and gaps) versus overlaps (e.g., intersections and clashes), degenerate shape features, abnormal size (e.g., too small or too large) elements, inconsistent orientations, and numerous other failure events that can be tested.

---

### 3.3.4 Use-Case #1.4: Data-Centric Exchange (Standardizing on Neutral Formats)

---

**Basic Idea/Setup.** The $N^2$ translation challenge of Use-Case #1.3 is usually addressed by standardizing on 'neutral' file formats (e.g., STEP, IGES, and other de facto standards such as STL).

In principle, this appears to provide some interoperability—only as rich as the neutral format's expressiveness—for all systems that can import/export such a neutral format. But it should be clear that this interoperability solution is subject to many of the limitations of the Use-Case #1.3. Additional difficulties arise when the interoperating systems A and B are communicating indirectly through the neutral format N and are unaware of the assumptions and internal implementation details made in the other system.

**Examples.** The PDES/STEP effort is the representative example for this use-case. It is a growing standard that aims to accommodate as much shape detail as most representation schemes in common usage can handle. The main challenges with STEP are its "flavor" diversity, which defeats the standardization purpose, and limited expressiveness (e.g., for specifying manufacturing tolerances). In addition to the STEP standard(s), de facto specialized standards exist in different application domains, including IGES for curve and surface data [66], STL, OBJ, and VRML for polygonal mesh

[20] This is sometimes referred to as the '$N^2$ problem', N being the number of CAx systems (see also Footnote 9).

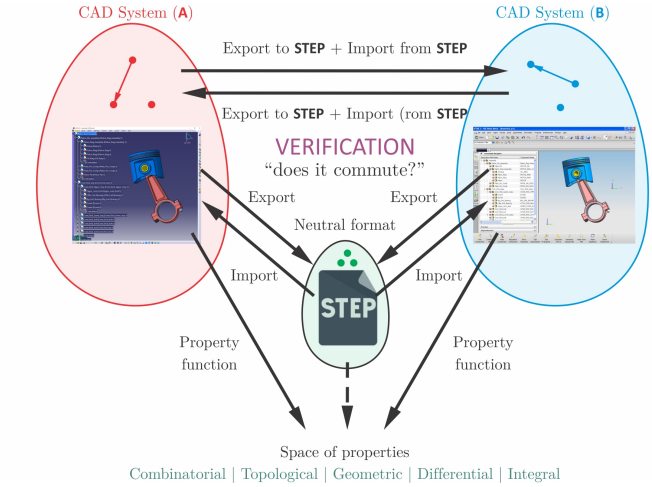



Figure 22: Use-Case #1.4: Representations (bullets) are converted via export/import to/from neutral file format (e.g., STEP), whose interchangeability can be verified by testing them against the properties that should be preserved. Interchangeability of algorithms (arrows) remains a challenge.

data, and many other application specific formats. The STL (based on triangular surface mesh) is particularly popular in many applications, including but not limited to 3D printing, because of its simplicity and wide applicability. If both A and B use mesh-based internal representations, STL can be used effectively without approximation (assuming consistent storage precisions). But when exchanging information between higher-order representations, STL is at best an approximate neutral representation to be used as an external reference with respect to which the quantitative properties can be measured (e.g., upper bounds on Hausdorff distance).[21]

Even simpler than STL (in both primitives and composition rules) are voxel maps. They provide a great device as a generic set of coordinate axis-aligned "rulers" that are externally addressable for measuring geometric properties. In addition, it provides a basis for geometric hashing as a special form of partitioning the modeling space into interchangeability classes, where two models that 'snap' to the same collection of voxels can be deemed interchangeable.

Many other example are conceivable, ranging from algebraic surfaces of various orders and discretizations to groupings of balls and point clouds. One important point to remember is that as neutral representations becomes simpler to use, they may also turn lossier, requiring additional assumptions to recover the lost information. This, in turn, makes it harder to provide an inverse map (i.e., import) as discussed below for separated verification for A $\rightleftarrows$ N and N $\rightleftarrows$ B. Although neutral formats are predominantly representation devices with no algorithmic content, it is conceivable that the standards can grow to accommodate algorithms, for example to accommodate procedurally defined 3D printed shapes.[22]

**Properties.** Similarly to the previous use-case, the properties are specified with the recognition of trade-offs between shape (i.e., combinatorial, topological, geometric, differential, and integral) properties exemplified at the beginning of this section. The main difference is that here, the prop-

[21] It is worthwhile emphasizing that if the STL data is used as an intermediate step—i.e., export from A to N and import from N to B—then the case belongs here to Use-Case #1.4. However, if the STL data is used solely as an external reference for measuring properties, but is left out of the system-to-system conversion—i.e., direct export/import via native file formats of A and B—then the case belongs to Use-Case #1.3 discussed earlier. The two scenarios have very distinct attributes and should not be confused.

[22] See also Use-Case #1.5 in which the standardization is on construction procedures, which are predominantly algorithmic.

erties must be computable purely from the neutral format's information content. In other words, the properties cannot depend on the information that is lost upon exporting to the neutral format, as expected. This immediately reveals the inherent limitation of this approach; namely, one's invariance guarantees can be only as good as their neutral format. If a property cannot be captured by N, it cannot be guaranteed to remain invariant either, as the information is channeled through N—to which we referred earlier as a "semantic bottleneck."

The ISO 10303 standard for product data representation and exchange [2] has grown significantly to include not only nominal shape specifications, but also design intent, tolerancing, manufacturing data, and more, with ongoing research efforts in each direction. This evolution of the neutral format effectively expands the space of available invariant properties but, intrinsically, remains incomplete as new material structures, fabrication processes, and technical challenges emerge. For example, geometric dimensioning and tolerancing (GD&T) standards are a key component of any neutral standard for description of mechanical components; in contrast, as of today, no neutral standards provide support for exchange of toleranced geometric data in the presence of modeling errors.

When simplified and approximate shape representations (e.g., STL meshes, VDB voxel maps, PCD point clouds, etc.) are used as the neutral format, the scope of properties that can be preserved are further limited. For instance, converting to STL results not only in a loss of the constraints that capture design intent and/or dimensioning and tolerancing, but also in a decay of the shape properties. In particular, piecewise linear approximations lead to geometric errors and loss of differential properties (e.g., curvature and $G^k-$continuity). Voxel maps do not preserve surface normal information altogether, point clouds also lose topological properties, and so on. Note that each of these simplifications or approximations are useful in their own limited capacity—determined by as many of the properties as they retain, but preservation of metric properties requires additional external references against which such properties can be measured.[23]

Similarly to the previous use-case, the property functions are computed via entirely different algorithms, and the burden of each function's correctness is on the authoring system. The same applies to the neutral format and its semantics for computing the properties from the exported data. Additionally, the properties computed by the two systems and on the neutral format must agree, i.e., exporting the data to the neutral format N (from A and B) followed by $\mu_{\mathsf{N}} : \mathsf{N} \to \mathrm{P}$ must give the same result as $\mu_{\mathsf{A}} : \mathsf{A} \to \mathrm{P}$ and $\mu_{\mathsf{B}} : \mathsf{B} \to \mathrm{P}$, respectively. For example, if STL is being used, and one cares to preserve a particular integral property (e.g., total volume or surface area inside a given bounding box), the polygonization step in export to STL must preserve that property, such that $\mu_{\mathsf{A}}$ and $\mu_{\mathsf{B}}$, computed on their respective internal representations, and $\mu_{\mathsf{N}}$, computed on the STL mesh, yield the same results—or agree up to a pre-specified tolerance.

Importantly, when computing the properties from N's semantics, one cannot use additional assumptions that are not part of the standard. For example, if the neutral format is voxel-based, any property that depends on the lost surface normals should not be taken as an invariant, unless the assumptions used in recovering the surface normals are made explicit in the specifications of the neutral format.

**Interoperability Map.** In this use-case the interoperability map is a compound process that can be broken down into 1) export from A to N, followed by 2) import from N to B—with or without interruption by third party healing/repair tools discussed in Use-Case #1.3.

Once again, developing the export/import utilities subjected to specified conditions on what is preserved at each step is solving the correspondence problem. If one is already given a legacy export/import solution without any such specifications, one encounters the harder characterization problem. In each case, preserved properties are the common part (i.e., intersection of) the set of properties preserved from A to N (i.e., export) and from N to B (i.e., import).

[23] Nevertheless, simplified representations themselves may serves as such external references (e.g., for Use-Case #1.3) even if they are not ideal for an intermediate step in the translation process.

**Verification.** The a posteriori verification problem can be significantly simplified if both systems can separately guarantee property-based interoperability (using the same external semantics for properties) with the neutral format. Specifically:

- Both systems' export/import functions to/from the neutral format are inverses of each other up to interchangeability with respect to the specified invariants; i.e., converting back-and-forth between each systems' internal representation and the neutral format keeps the properties invariant, no matter how many times it is repeated.
- The common part (i.e., intersection of) invariant properties in A $\rightleftarrows$ N and N $\rightleftarrows$ B conversions covers the set of properties that one desires for A $\rightleftarrows$ B interoperability.

In other words, the transitivity of interchangeability with the neutral format allows one to decompose the verification problem to independent ones that do not need collaboration between the developers of A and B, as long as they both can work with the common standard.

For example, if A and B exchange CAD data via STEP, as A $\rightleftarrows$ STEP $\rightleftarrows$ B (arrows denoting export/import), to verify that the composite export+import maps A $\rightleftarrows$ B preserve a given property, it is sufficient to verify that each of A $\rightleftarrows$ STEP and STEP $\rightleftarrows$ B preserve the same property. This, in turn, can be done by back-and-forth testing the export/import utilities of each system individually against STEP—perhaps repeated in multiple round trips to amplify the errors for a more revealing analysis. The transformations may or may not change/degrade the model after enough iterations; however, the properties must not change.

**Advantages.** The main advantage of using neutral file formats is to centralize the interoperability through a common standard and reduce the number of translators from quadratic in Use-Case #1.3 to linear. Every time a new system or representation scheme emerges, it need not interoperate with all previous systems, but need only support export/import to/from the neutral format. Of course it takes more than simply providing such functionalities, and the two-way verification with N is in order with respect to specified invariants. Additional advantages include:

- It is generally easier to figure out what properties are being preserved, in contrast to the direct translation scenario in Use-Case #1.3, when they depend solely on the neutral standard's semantics—which are usually better documented and widely available on the public domain.
- As discussed earlier, verification can be decomposed to two separate tasks, performed in isolation between each system and the neutral standard, without requiring the system developers to work together for a direct system-to-system verification.
- The third party neutral format, if used properly, can provide an external reference for transitive interchangeability with respect to quantitative properties, capping error accumulation.

**Drawbacks.** Despite the appeal of simplified and approximate representations as a neutral exchange standards, they offer few guarantees, preserve few properties, and do not enforce transitivity under multiple exchanges. The resulting information loss makes the aforementioned verification process (based on inverting export to import) harder to achieve, as mentioned earlier.

At the other extreme, in an effort to accommodate more information and preserve more properties, the highly expressive STEP standard has turned into a superset of "flavors" of different formats with subtle differences. No single flavor is sufficient for all purposes—as is the case with systems and representation schemes—and an attempt to capture the difference by allowing varieties defeats the standardization purpose.

It is difficult to incorporate tolerances to standard formats to formalize interchangeability in the presence of errors, which may explain why geometric accuracy and tolerance information continue to be missing from STEP. The challenge is to incorporate externally addressable references that would

allow to 'hash' different representations into interchangeability classes. The voxel maps appear particularly attractive for this purpose, though other representations may be used as well. Until this difficulty is resolved, having export/import functions that are inverse to each other up to property-based interchangeability cannot be taken for granted due to round-off or approximation errors. This in turn implies that data-centric translation is not likely to achieve satisfactory transitivity and mitigate model degradation.

Last but not least, even as the neutral standards mature and keep growing, they are often based on premature decisions that may be incompatible with new representations and algorithms, stifling the innovation.

**Best Practices.** The choice of a neutral format boils down to a trade-off between expressive power for carrying more properties and simplicity to develop translators than can preserve them. This can be viewed as a trade-off between the scope of characterization and feasibility of establishing correspondence, respectively. In essence, the neutral format must be simple enough to be usable by interoperating parties without confusions about semantics, inconsistent implicit assumptions (e.g., approximating properties that are lost)—but not too simple, such that it misses important semantic information.

It is a common pitfall to assume highly expressive standards (e.g, STEP) are lossless, when they are not. One must be aware of the limitations of the chosen format and whether it is sufficient for the purpose at hand—e.g., voxel maps might be perfectly adequate for preliminary synthesis and topology optimization iterations, but may not be expressive enough for the final design.

One should be wary of flavor diversity of standards and the resulting semantic inconsistencies. It is best to be upfront and consistent on which flavors are used throughout the workflow. Having an explicit specification of invariants facilitates re-thinking the choice of flavors in future or interoperating with other flavors. It is best to go with the bare minimum that supports the properties.

When it comes to developing the standards for neutral file formats, it is best to shift the focus to the properties one cares to preserve, rather than arbitrary decisions based on syntactic simplicity, expressive generality, etc. It is generally ill-advised to enlarge the standard to support as much representational variety as possible without worrying about semantic inconsistencies.

---

### 3.3.5 Use-Case #1.5: Generic Model Exchange (Procedural or Declarative Recipes)

---

**Basic Idea/Setup.** To mitigate the difficulties in Use-Cases #1.2, #1.3 and 1.4, stemming from incompatible interpretations of fully instantiated CAD models in different systems, an alternative solution is to standardize on the construction recipes for instantiating CAD representations, instead of standardizing on the representations themselves. Such construction recipes are often called 'generic' models and include both procedural (e.g., parametric, generative, or constructive) and declarative (e.g., constraint-based) descriptions of CAD models.

The generic models are largely symbolic and compact structures that are formally defined by external syntax and semantics. When two systems A and B with different representations and/or algorithms, agree on such a generic modeling language L, their interoperability is reduced to each system's ability to interpret the generic models—with or without internally producing fully instantiated CAD representations. Examples of the operations and relations based on which generic models can be built are Boolean operations, Minkowski operations, sweeps/unsweeps, offsets/blends, motions/deformations, dimensioning and assembly constraints, and a full range of application-specific feature-based representations.[24]

[24] Philosophically, rather than export/import to/from a neutral format for the 'state', this scenario aims to

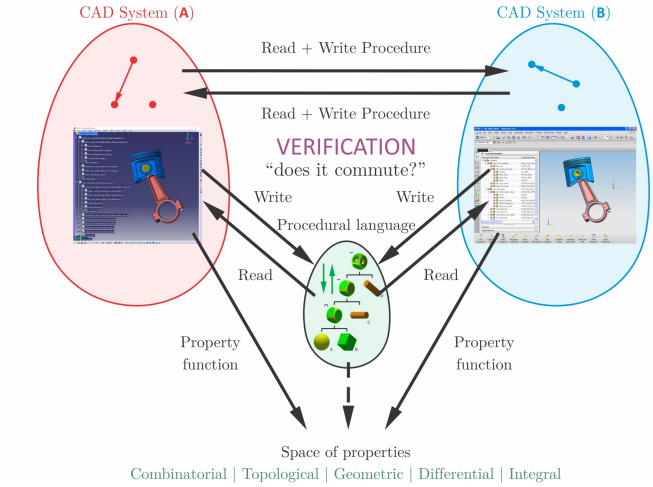



Figure 23: Use-Case #1.5: Procedural recipes are interchangeable with algorithms (arrows) that construct the representations (bullets) separately in each system. The responsibility of evaluating them correctly, in compliance with the common semantics, is assumed by the systems.

**Examples.** The constructive solid geometry (CSG) representation scheme for solid modeling is a classic example of a basic procedural language, whose operations (namely, Boolean operations and rigid motions) have universal semantics. Thus if they also agree on a finite set of primitives and how they are parameterized,[25] A and B are guaranteed in advance to produce interchangeable representations—an example of a priori verification.

This approach extends in a straightforward fashion to many other operations, transformations and constraints. Unfortunately, exchange of generic models becomes problematic for more complex operations whose semantics depend on evaluation or interpretation of other simpler operations. Notable examples include edge blending operations that may be defined in several (incompatible and order-dependent) ways. Feature instantiation operations often require references to previously instantiated edges and faces; unfortunately, not only these edges and faces are subject to numerical errors, but also these references may not be persistent under different algorithms or even under change of parameters in the same system.

Below, for the purpose of this discussion, we assume that generic models are described in the common language L with unambiguous semantics, but in practice a number of fundamental problems remain unsolved including persistent naming, robust computing (e.g., for intersections), and consistent semantics (e.g., for blending) mentioned above.[26]

It is worthwhile also pointing to a more practical application of this use-case scenario, which is for shape synthesis and optimization. A major benefit of exchanging parameterized construction procedures instead of the final result is that the information can be treated as a *family* of shapes, rather than a single one. Thus it supports automation through iterative synthesis+analysis, enables encoding design intent into the procedures, facilitates inferring gradient information, etc.

**Properties.** The reasoning must be clear by now: the preserved shape (e.g., combinatorial, topological, geometric, differential, and integral) properties must be computable from the generic model

write/read to/from a common language for the 'process' that generates it.

[25] One can argue that agreeing on geometric primitives (e.g., semialgebraic halfspaces) is also subject to nuances in precision semantics (e.g., round-up errors). Nevertheless, defining interchangeability for primitives is a much simpler task than it is in general for evaluated representations in data-centric methods (Use-Cases #1.3 and 1.4).

[26] There is a rich body of literature on feature-based solid modeling [27, 14, 62, 65, 54, 9, 55, 67, 28].

semantics alone. Any system-specific assumption or decision that cannot be inferred from the procedural or declarative content should not be used to define properties. Accordingly, the properties are restricted to those fully specified by the language semantics and cannot be extended to those that are left to the systems' discretion.

Importantly, unlike previous use-cases, many shape constraints that capture the design intent (e.g., symmetry, parallelism, coaxiality, etc.) as well as manufacturing intent (e.g., manufacturable features, dimensioning and tolerancing, etc.) are preserved and transferred via predominantly symbolic structures, minimizing or eliminating their susceptibility to numerical errors.

Similarly to the previous use-case, the property functions are computed via entirely different algorithms, and the burden of each function's correctness is on the authoring system. The same applies to the procedural language and its semantics for computing the properties from the common procedure. Importantly, a property function defined on the procedural representation/program $\mu_{\mathsf{L}} : \mathsf{L} \rightarrow \mathrm{P}$ precisely specifies how the property is computed from the program's static structure alone, independently of which system will be eventually compiling it.

For example, take two CAD systems, based on any internal representation scheme (e.g., B-rep, hybrid B-rep+CSG, etc.) that exchange information via CSG trees, caring to preserve a particular integral property (e.g., total volume or surface area inside a given bounding box). Then computing that property over either system's evaluated representation—i.e., after separately instantiating the same CSG tree into their own internal data structures—must agree, meaning that it is possible to compute it directly on the CSG tree in the first place. If the two cannot agree precisely, one can specify tolerances as upper bounds to the deviation of the properties computed in each system from that of the exact property computed on the CSG tree (serving as the external reference).

**Interoperability Map.** Conceptually, the interoperability map is a compound process that can be broken down into 1) write from A to L, followed by 2) read from L to B. The write/read in this case are not necessarily file export/import (unlike file-centric methods in Use-Cases #1.3 and 1.4), though files are viable containers to use among other possibilities (e.g., real-time message passing). Here, the writing refers to the process by which the procedural and/or constraint-based recipe is retrieved from the sending system (i.e., A), and the reading refers to the instantiation of the recipe by the receiving system (i.e., B). These can be done in a number of ways:

- Each system can choose to base its internal computing on the procedure itself, postponing its evaluation (e.g., 'lazy' evaluation methods). The advantage of minimizing 'side-effects' in this case is could be overshadowed by consequent performance limitations.
- Each system can choose to maintain a construction history at all times alongside its evaluated internal representation (if there is one). The obvious challenge is that of maintaining internal consistency (i.e., interchangeability) for such 'hybrid' representation schemes.
- The alternative is to instantiate the procedural recipe upon reading, and attempt to reconstruct it on-demand upon writing by re-parameterization of the shape (i.e., shape factorization). As with most feature recognition/retrival problems, it is unlikely that this process can be fully automated in a reliable fashion, partly because there is no unique answer in general.

In general, the procedural input can be fully evaluated, partially evaluated, maintained alongside evaluated into a hybrid representation, etc. In all cases, the interoperability problem is shifted to each system's own assurance of internal consistency (e.g. correct boundary evaluation from CSG).

**Verification.** Regardless of how the systems decide to go about write/read of generic model recipes to solve the correspondence problem mentioned above, the verification problem can be decomposed into two separate tasks for $\mathsf{A} \rightleftarrows \mathsf{L}$ and $\mathsf{L} \rightleftarrows \mathsf{B}$. Verifying that these tasks are inverse of each other up to interchangeability implies that repeatedly applying them back-and-forth to an arbitrary model will leave the specified properties intact.

**Advantages.** It is appealing to standardize the semantics for data exchange based on higher-level, largely symbolic feature-based models than, for example, neutral file formats (Use-Case #1.4). The fact that they are unevaluated—thus do not usually include computed numerical values—make procedural models less affected by the robustness issues and model degradation. The validity and correctness are guaranteed by design; in this sense, constitutes the "cleanest" form of interoperability—or illusion thereof, when formal semantics issues are not fully resolved.

The unique feature of this scenario is that it allows exchanging not only single models, but also parametric families of models, in which a lot more information is encoded including but not limited to design intent, manufacturing intent, and gradient information (e.g., sensitivities for shape and topology optimization).

**Drawbacks.** The main drawback of this approach—which can easily go unnoticed—is the lack of any guarantees for semantic consistency. The systems assume the responsibility of enforcing correct semantics internally. The way each one interprets and instantiates each instruction or feature in the procedural recipe (e.g., what a 'hole' or 'pocket' mean to A vs. B) is taken for granted. One can attempt to standardize virtually all significant features, their attachments, and placement conditions using various ontologies—e.g., classifying features to generative, modifying, and referencing features [13, 12]. However, correct and consistent interpretation of such definitions requires solving the open problems of order-dependence, persistent naming, and robustness. These challenges may be compounded by the practical difficulties of ensuring that procedure write/read in different systems are in fact inverses of each other up to interchangeability.

**Best Practices.** Generic model-based interoperability is well positioned to circumnavigate the need for externally representing a 'master' model—e.g., external geometric references anchored to control error accumulation in Use-Cases #1.3 and 1.4. The price is a semantic "trap:" the language must provide construction semantics to the last possible detail, *and* each system is responsible for enforcing them internally. One should not assume that syntactic interoperability (e.g., ability to write/read the language L and instantiate it to always valid structures) on the part of the systems is sufficient.

The method works best for universally accepted concepts (e.g., Boolean or Minkowski operations) that are representation agnostic, without any controversy on math/semantics. For example, a global blend defined as a morphological opening/closing with a ball—in turn, defined as a sequence of morphological dilation/erosion (i.e., Minkowski operations)—has well-defined representation-independent semantics. A local blend (e.g., of a particular edge or vertex), on the other hand, references internal representation elements and is susceptible to interoperability issues due to ordering rules, persistent naming, and robustness.

Extending the language without restraint—using arbitrary new definitions, assumptions, and constraints to enhance its expressive power—is ill-advised. Every such decision should be made in observance of what it takes to fully specify the semantics, handling complicated scenarios (e.g., overlapping blends), ordering rules for constraint resolution, etc. These rules must be documented for both systems to adhere to.

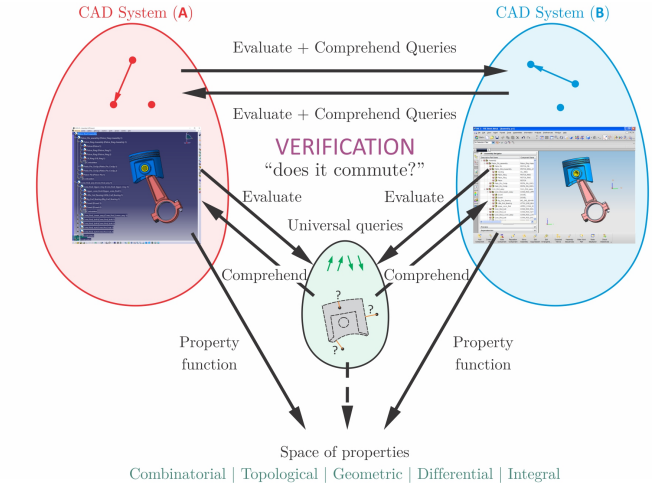



Figure 24: Use-Case #1.6: Queries are simple functions that from the building blocks for the interchangeable algorithms (arrows) in each system, from which the representations (bullets) can be reconstructed, up to interchangeability, by evaluation+comprehension.

### 3.3.6 Use-Case #1.6: Query-Based Exchange (Standardizing on Functional Queries)

**Basic Idea/Setup.** Query-based interoperability is radically different from the other use-case scenarios in that it aims to bypass the CAD model exchange problem altogether. Unlike data-centric or procedural methods (Use-Cases #1.3–1.5), the model may never need to be transferred in its entirety (e.g., via export/import files or write/read programs). Moreover, unlike Use-Cases #1.1 and #1.2, A and B may have dramatically different representations and algorithms. Rather, the systems retain their own copies separately at all times, and communicate small packets of information only when the need arises (i.e., via send/receive queries).

The 'queries' are computable functions whose semantics are specified with respect to a common reference. They depend on the nature of the tasks at hand, for which information about the exchanged model is needed [30]. Similar to the property functions, the burden of their correct implementations in compliance with the common semantics is assumed by A and B separately.

**Examples.** Consider a scenario in which A and B exchange information via points only, i.e., no higher-dimensional simplices (e.g., edges or faces) are exchanged. All that passes through the system boundaries are 3D coordinates of a finite number of query points—with respect to a common Cartesian coordinate system—sampled (at once or adaptively) in the interior or on the boundary of the shape in question. If the systems also agree on the semantics of a few other basic questions about the bounding box, types and number of primitives, internal rules and constraints for parameterizing the primitives (e.g., maximum polynomial degree) and other information, they could (but do not have to) reconstruct one another's models after a *finite* number of exchanged queries up to any desired fidelity, accuracy, and resolution—or more generally, up to an arbitrary specification of property-based interchangeability relation.

This basic sampling scenario can be extended to sampling higher-dimensional spatial elements, e.g., sending/receiving simplices or simplicial complexes or performing set-membership classification

(SMC) [69], computing distance and integral properties, querying spatial intersection measures (e.g., asking "what fraction of this voxel is inside?"), and so on. Increasingly complex queries corresponding to nontrivial geometric tasks may be constructed from simpler queries.

**Properties.** The properties that one cares to preserve must be computable from the queries alone—or more precisely, from a *finite* sequence of queries. This observation blurs the distinction between the properties and queries. The queries (and hence the properties) range from global attributes (e.g., dimension and bounding box) to local questions (e.g., membership and distance). They can request combinatorial, topological, geometric, differential, and integral information, as long as the underlying mathematical concepts are universal, non-controversial, representation-agnostic, or otherwise recognized and agreed-upon by both systems.

Under common assumptions (e.g., on bounding box size, maximum resolution, and minimum feature size), these point membership and distance-to-boundary queries are sufficient to compute most practically significant geometric and topological properties after a finite number of queries, without explicit exchange of representations or files.

The query object (i.e., 'candidate' set) need not be a point either; for example, B can send simple higher-dimensional elements (e.g., line segments, polygons, and polyhedra) to query average measures of their intersection against the solid (i.e., 'reference' set) in A. Alternatively, B can ask A to generate its own set of points (e.g., sampled over a surface), and so on. See [30] for more details.

Obviously, the results computed by property functions must not depend on system-specific assumptions, and must be obtainable (in principle) from the common semantics for queries—which may be defined analytically, computationally, or empirically. For example, one can choose the property function as the collective response of a system to a large number of PMC tests (e.g., sampled on a uniform grid in a bounding box). As long as the point cloud and the meaning of PMC is the same for both systems, it is a valid interchangeability certificate between A and B. The same can be done using distance queries (with or without tolerances) on a common grid, measure queries over a common voxel map, etc.

**Interoperability Map.** Unlike the previous use-cases, solving the correspondence problem should not be viewed as an attempt to develop an explicit data conversion process. Rather, the goal is to understand the process by which the receiving system (i.e., B) *can* reconstruct the model in the sending system (i.e., A) in its own representation scheme, even though it may never do so in practice.

In other words, the correspondence problem in this use-case is to establish an implicit association between A and B's internal representations and algorithms, while allowing each system keep them hidden and query them on-demand for partial information.[27]

**Verification.** The verification task reduces to certifying the correspondence between all primitive queries and their composition rules. While the task is nontrivial, it is clearly defined, assuming that common semantics exist as a result of characterization. Furthermore, with common reference semantics in place, verification of queries becomes a question of compliance and certification of the primitive queries in the individual systems A and B with respect to that semantics. Once such compliance is established, the correspondence is also verified for every property that can be described by a valid composition of the primitive queries.

**Advantages.** The main advantage of the query-based approach is that it lends itself well, almost by design, to the property-based interchangeability paradigm. The queries serve as a basis to span both property functions and interoperability map (e.g., by purely functional composition).

[27] An important question is, what is the minimal set of (preferably "simple") queries, along with a set of basic composition rules that suffice for establishing such a correspondence?

Consequently, if the information exchange is channeled through the queries alone, without any exceptions or "hacks," the query-based interoperability map preserves the properties (by design).

The approach can support variety of formal semantic models. For example, one could extend the abstraction of 'crisp' solid models, with binary membership semantics and deterministic queries, to 'fuzzy' solid models with real-valued membership grades, density functions, and probabilistic queries [75, 44, 76] without changing the main framework.

Furthermore, solving the correspondence problem in this use-case does not require access to source-code or collaborative development between A and B. This has significant economic and strategic implications, as query-based information exchange respects proprietary internal structures. Moreover, it abides by the same modularity, data hiding, and encapsulation principles that have made object-oriented programming [45] a great success in software engineering. These principles allow fine-granularity, agility (in adapting/extending to new abstractions), and support for the development of open and composable software architectures. The query-based approach to interoperability supports building resilient workflows by self-correcting, interactive, and adaptive data exchange. Errors are grounded rather than accumulated by constant re-iterations.

**Drawbacks.** Despite the great promise of query-based interoperability, its power, formal properties, and practical limitations remain to be understood. Here are a few caveats to keep in mind:

- The queries must be formally defined with respect to standardized semantics that are mutually agreed-upon by all interoperating systems. This task is not trivial and has not been attempted. The interoperability guarantee extends only as far as the standard. For example, if the standard does not deal with errors and accuracy, the interoperability guarantee would not extend to properties that depend on them.
- Similar to the procedural methods (Use-Case #1.5), the burden of implementing and certifying correctness of queries and their compliance with the standard lies with the authoring system—though the simplicity and universality of basic queries (e.g., membership and distance) makes it easier to do so in contrast to complex procedural features and operations.
- What is a good set of primitive queries? In principle, keeping the interfaces as small as possible is what distinguishes queries from large data exports in data-centric methods (Use-Cases #1.3 and 1.4) and their complications. However, the modularity and data hiding benefits that come with smaller interfaces are sometimes at the expense of computational performance.[28]
- The approach is fairly recent and is not supported by the existing systems and infrastructures. Its wide application requires reformulating many applications and algorithms in terms of queries, which is nontrivial.

**Best Practices.** The benefits of query-based approaches over data-centric methods are realized only when the interfaces are as small as possible. Many light and simple queries are preferable to few large exchanges with complicated semantics. The approach works better in practice when the query input/output data are based on simple types—e.g., real-valued coordinates and distances—rather than complex combinatorial structures. The larger and messier the queries become, the more may they become susceptible to the limitations of data-centric or procedural exchange (Use-Cases #1.3–1.5). It is best to take a "minimalistic" approach and add queries only when absolutely necessary, along with their assumptions and semantics (e.g. as pre-/post-conditions [45]).

Hacks and shortcuts to improve the performance are ill-advised from an interoperability perspective. If keeping the queries at the bare minimum would result in prohibitively slow algorithms, one can "batch" them together into larger queries.

[28] The rapid growth in high-performance computing (HPC) offsets this disadvantage to a great extent. As long as the queries used fairly independently with little synchronization overhead, the query-based approach is well-suited for massive parallelization (e.g., on GPUs).

**Summary of CAD-CAD Data Exchange Scenarios.** Since the inception of solid modeling, the CAD-CAD data exchange has been approached as a problem in model-based interoperability, starting from a premise of informational completeness, where the ultimate goal is to preserve the shape integrity as much as possible. Our analysis reveals that the only true solution that achieves model-based interoperability is that of Use-Case #1.1 which eliminates the interoperability bottleneck by adopting the same system across all applications. However, in most realistic scenarios (e.g., Use-Cases #1.2–1.6), it is unlikely to be able to preserve everything; trade-offs need to be made depending on which properties one cares the most about. This is the essence of the property-based interoperability approach.

In general, it is conceivable that best performance is achievable by a combination of the data-centric, procedural, and query-based methods, as long as one is aware of the pros and cons. In particular, if A and B have commonalities (e.g. in their combinatorial data structures), it may make more sense to exploit it, at the expense of the drawbacks discussed in Use-Case #1.2. If any form of representation conversion is added to the mix, one should be aware of the drawbacks discussed in Use-Cases #1.3 and 1.4.

# 4 Use-Case Scenarios: CAD-CAx Interoperability

CAD-CAx interoperability usually appears as system integration: the CAD system is integrated into the CAx system (e.g., CAE and CAM among others) in a manner that allows the CAx system to perform tasks relevant to a particular type of engineering activity. The shape information is provided by the CAD system, and (often partially) utilized within the CAx system, using the same or different representation schemes. The six use-case scenarios discussed for geometric interoperability in Section 3 for CAD-CAD data exchange apply to this mode of CAD-CAx interoperability. Rather that repeating their properties, we focus on the shape-dependent tasks that are required by the CAx activities and normally are *not* directly supported by CAD systems.

**Surrogate Properties.** Shape properties plays a dominant role in CAx activities as well. In addition to some—but not necessarily all—of the shape (i.e., combinatorial, topological, geometric, differential, and integral) properties that need to be preserved, other properties may be of interest as well, depending on the CAx application. However, these properties may not be known a priori, in which case the shape properties are used as a *surrogate* to certify interchangeability, as elaborated in Section 2.2.1. This is due to at least two reasons:

- The CAD system A does not necessarily have the capacity of representing and interpreting the additional semantics processed by the CAx system B (e.g., physical fields, fabrication plans, and material structures). Thus its property function $\mu_{\mathsf{A}} : \mathsf{A} \to \mathrm{P}$ cannot be expected to capture such properties.
- Even if the capacity to represent such information is added to the CAD system A (e.g., for visualization purposes), there may not exist a standardized external base of comparison to certify the interchangeability of the exchanged non-geometric information between A and B.

In the following sections, we shall discuss these issues in the context of two CAD-CAx integrations: CAD-CAE and CAD-CAM.

**Trade-Offs and Prioritization.** In many CAD-CAD data exchange scenarios between similar systems, the ultimate goal is to make the transfer as lossless as possible, preserving as many shape properties that the two interoperating systems' differences allow. This is aspired without a particular downstream application in mind, which makes it harder to pass judgements about which properties matter more than others to decide the trade-offs. In contrast, CAD-CAx integration provides a "context" to make such decisions from a pragmatic viewpoint:

- CAD-CAE integration (Sections 4.1 and 4.2) cares about correct physical simulation, which can be broken down to a few fundamental operations, e.g., computing volume and surface integrals, enforcing boundary conditions, assembling/solving linear systems, and so on.
- CAD-CAM integration (Sections 4.3 and 4.4) cares about correct fabrication planning, which can be broken down to a few fundamental operations, e.g., identifying unit actions, high-level sequence planning, tool-path generation, enforcing tolerance specifications, and so on.

In each CAx activity, it is not always practical to preserve all shape properties due to differences in representation semantics. Thus preserving the subset of shape properties that influence the above operations the most, should be prioritized in each integration scenario.

CAD and other CAx systems usually serve different purposes that depend on different forms of computations. These computations often lend themselves better to some representations and algorithms than others. Nonetheless, these representations and algorithms must agree on common references for the meaning of these computations.

## 4.1 CAD-CAE Integration Properties

For the purpose of this discussion, we assume that the CAE system's task is to solve a well-posed initial/boundary value problem defined over the geometric model provided by the CAD system. Generally speaking, the CAE analysis

- accepts the CAD model, which usually includes descriptions of shape (geometry and topology), material properties, and assembly relationships;
- accepts the initial/boundary conditions, which are associated with the CAD model's boundaries and interfaces with other elements; and
- computes the physical response, typically in the form of spatio-temporal fields which are associated with (and may or may not be communicated) to the CAD model.

We restrict ourselves to the most common case of finite element analysis (FEA), in which the tasks are usually achieved as follows:

1. Assume that the solution is represented by a linear combination of basis functions whose supports cover the CAD model. The linear combination of the basis functions define a space of admissible solutions, meaning that they satisfy the imposed initial/boundary conditions.
2. The coefficients of the basis functions are selected to minimize some functional determined by the weak form of the initial/boundary value problem. This tasks requires numerical integration of (functions of) basis functions and their derivatives over the supports of basis functions that are contained inside the domain. These integrals populate a linear system of equations that is solved for the coefficients of the basis functions.
3. The solution expressed as a linear combination of basis functions is associated with (points of) the CAD model, for example, as a deformation, temperature, stress, etc.

Thus, in addition to the usual CAD tasks, a CAE system must support the following tasks. Each task's successful completion require preserving certain properties, discussed below:

- **Task 1 (a):** Associate prescribed initial/boundary conditions with portions of the CAD model's boundary, at discrete time instants or over continuum time intervals.
  - **Property**: (Portions of) the CAD model's boundary and attributes prescribed over them.
  - **Property-based Interchangeability:** Subsets of the boundary (e.g., surface patches, and in some cases curve segments and points) that are decorated in the CAD model with annotations pertaining to physical behavior must match with specifications of the physical analysis problem in terms of initial and boundary conditions to be enforced.
- **Task 1 (b):** Associate prescribed internal/external regions of the CAD model with material properties and body effects, at discrete time instants or over continuum time intervals.
  - **Property**: (Portions of) the CAD model's internal or external regions and attributes prescribed over them.
  - **Property-based Interchangeability:** Subsets of the interior/exterior (e.g., volumetric cells, and in some cases lower-dimensional elements) that are decorated in the CAD model with annotations pertaining to physical behavior must match with specifications of the physical analysis problem in terms of material properties (e.g., stiffness, conductivity, and so on) and body effects (e.g., gravity, electromagnetic fields, and so on).

Interchangeability with respect to boundary conditions does not imply that the supports of basis functions need to conform to the boundary, though that could simplify their enforcement by assigning them to individual finite elements. The ultimate goal is to ensure that the solution field, represented in a linear basis with conforming or non-conforming supports, is admissible, i.e., interpolates the constrained valued on the boundary:

- **Task 2:** Enforce prescribed initial/boundary conditions (typically for Dirichlet conditions) on linear combinations of basis functions.
  - **Property**: Admissibility of the spatio-temporal field(s).
  - **Property-based Interchangeability:** The spatio-temporal solution defined over the CAE model as a linear combination of the basis functions must take on the specified values and derivatives at specified locations/times, at least within a well-defined tolerance with respect to an external reference.
- **Task 3 (a):** Perform volumetric numerical integration over the supports of basis functions contained inside the CAD model.
  - **Property**: Volume integrals computed over a finite number of bounded domains.
  - **Property-based Interchangeability:** Computing the integrals over a finite number of bounded domains must match between CAD and CAE representations of geometric model (if different), at least up to some approximation. The domains are typically described as the intersection of support functions with the geometric model.
- **Task 3 (b):** Perform surface numerical integration over portions of the boundary (typically for Neumann boundary conditions).
  - **Property**: Surface integrals computed over a finite number of bounded domains.
  - **Property-based Interchangeability:** Computing the integrals over a finite number of bounded domains must match between CAD and CAE representations of geometric model (if different), at least up to some approximation. The domains are the same as those deemed interchangeable in Task 1 (a); namely, surface patches on the boundary, and in some cases (e.g., impact forces) curve segments and points (degenerate integral).

If the supports of basis functions conform to the boundary, the volume and surface integrals become simplified, since the integral domains match with the supports of basis functions (i.e., 3D mesh cells) and their boundaries (i.e., 2D mesh faces). For non-conforming supports, such as those common in meshless/meshfree analysis methods, the volume integrals over the supports that partially intersect the geometric model near its boundary must account for the local geometry in one way or another, while surface integrals are typically handled by surface meshing (e.g., triangulation).

- **Task 4:** (Optional) Associate the solution computed by the CAE system with the geometric model authored in the CAD system (e.g., visualization of fields or deformation).
  - **Property**: Consistency of the domain and/or boundary parameterizations.
  - **Property-based Interchangeability:** The analysis results are mapped back onto the CAD model, often by re-parameterizing the internal/external domains or boundary surfaces and curves. The parametrizations need to match in terms of spatial coordinates at least up to some approximation (e.g., depending on visualization resolution).

Re-parametrization is particularly challenging near the boundary, for both approximately conforming and non-conforming supports of basis functions.

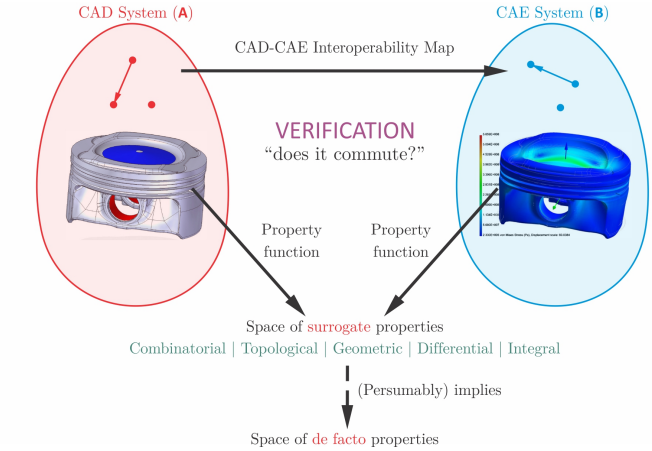



Figure 25: CAD-CAE interoperability verified up to invariance of surrogate properties.

To be clear, CAD-CAE integration may involve other tasks, such as computation of field sensitivities with respect to the CAD design variables, but we will limit our discussion in this document to interoperability with respect to the above tasks and properties only.

Depending on the mechanism for CAD-CAE interoperability and each system's representation schemes, these tasks may be challenging to varying degrees. For example, Task 1 (a) is trivial in any CAE system that supports references to the solid model's boundaries (e.g., B-rep surfaces), though persistent referencing can become challenging in situations such as changing parametric models. Task 3 (a) is relatively straightforward if the CAD system supports surface triangulation; Task 4 is supported by any CAD system that has access to shading and texturing algorithms. The interoperability challenges in these tasks are reduced to dealing with triangulation, shading, and texturing inaccuracies. Tasks 2 and 3 are more challenging in general.

## 4.2 Use-Case Schema #2: CAD-CAE Integration

By analogy to the shape data exchange scenarios in Section 3 (Use-Case Schema #1), here we present 3 out of 6 conceivable scenarios for CAD-CAE integration:

1. **Use-Case #2.2:** standardizes on a common basis to represent both geometric domains and spatio-temporal fields. Depending on the discretization scheme, it facilitates performing some of the above tasks and preserving shape properties without conversion. However, either or both systems become limited in their capabilities and performance as different representations and algorithms are suitable for CAD and CAE computations.

2. **Use-Case #2.4:** relies on representation conversion, and encompasses mesh generation with or without simplification, de-featuring, mesh healing/repair, and any other preprocessing. The CAD system can independently operate on representations that fit modeling activities best, as long as it supports meshing that suits CAE activities. However, meshing is hard to automate and comes with no guarantees without additional processing, which, in turn, may compromise shape properties pertinent to the aforementioned analysis tasks.

3. **Use-Case #2.6:** relies on complete delegation of geometric computations required by the CAE system to the CAD system. The global geometric information are never explicitly represented in the CAE system—or minimized to a bare minimum. Rather, the information is queried locally and lazily when needed by the aforementioned analysis tasks.

### 4.2.1 Use-Case #2.2: Standardizing on Representations (with Different Algorithms)

**Basic Idea/Setup.** The setup is similar to those of Use-Cases #1.1 or 1.2. The CAD system (i.e., A) and the CAE system (i.e., B) use the same representations (i.e., $\mathsf{M}_\mathsf{A} = \mathsf{M}_\mathsf{B}$), or at least the same core data structures, but differ in the algorithms (i.e., $\mathsf{F}_\mathsf{A} \neq \mathsf{F}_\mathsf{B}$)—namely, $\mathsf{F}_\mathsf{A}$ is a collection of algorithms that support geometric modeling operations such as Boolean operations, free-form surface editions, and so on, while $\mathsf{F}_\mathsf{B}$ is a collection of algorithms to support physical simulation, including integral computations, solving linear systems, and so on.

**Examples and Interoperability Problems.** Common representations used to support both CAD and CAE for "seamless" integration (i.e., no *conversion*) are voxel maps, polyhedral mesh, and higher-order curvilinear mesh.

Voxel-based representations are popular in topology optimization (e.g., SIMP method) and biomedical applications (e.g., 3D imaging), and are gaining popularity for visualization and geometric modeling (e.g., OpenVDB). Voxelization provides the simplest basis for FEA analysis, as the uniformity of the support functions allows assembly-free formulation and solution of the linear systems of equations. Despite being ideal for CAE, it is less so for CAD due to nonsmooth boundaries ("chessboard" patterns) that fall short in representing functional surfaces, lack of semantics for normal/curvature information, lack of closure under rotations, and other issues.

Polyhedral mesh-based representations dominate the FEA software systems, as they facilitate enforcing Dirichlet and Neumann boundary conditions, computing volume and surface integrals (Tasks 2 and 3), and assembling the linear system of equations for CAE, while providing some degree of modeling freedom for CAD. Despite several early attempts, finite element mesh is too restrictive as a primary representation for CAD since geometric operations ranging from Boolean operations to sweeps and projections become nontrivial.

Isogeometric analysis emerged from the recognition of a need for a common basis to represent both free-form geometric models in CAD and solution fields in CAE (e.g., using NURBS). For these higher-order finite element representations, enforcing Dirichlet boundary conditions is complicated and can be done in different ways with tradeoffs (e.g., 'strong' versus 'weak' enforcement methods) (Task 2). Neumann boundary conditions are satisfied in a similar fashion as in standard FEA. Computing volume and surface integrals require mapping to the parametric space and depend on more complex quadrature rules (Task 3). Although it has been largely successful for bivariate B-splines (e.g., analyzing shells) many problems remain to be solved for trivariate B-splines, trimmed surfaces, and so on. An additional benefit of these methods is that mapping the results back to CAD (Task 4) is particularly easy as re-parameterization is not needed. Since both CAD and CAE use the same geometric basis in this use-case scenario, shape-related properties remain invariant with the representation. The correspondence and characterization become trivial, and verification is guaranteed a priori. Of course, the elimination of the interoperability problem as such comes at an additional cost of case-specific challenges with making the same representation work for both CAD and CAE applications.

### 4.2.2 Use-Case #2.4: Data-Centric Integration (Direct or Indirect Data Translation)

**Basic Idea/Setup.** The setup is similar to that of Use-Cases #1.3 and 1.4. Unlike the previous use-case, the CAD system (i.e., A) and the CAE system (i.e., B) use dramatically different representations (i.e., $\mathsf{M}_\mathsf{A} \neq \mathsf{M}_\mathsf{B}$) and algorithms (i.e., $\mathsf{F}_\mathsf{A} \neq \mathsf{F}_\mathsf{B}$), accepting the reality that it is difficult to make a single common representation scheme support both activities without a compromise.

The CAD representation is first exported to a different (and often approximate) representation that lends itself better to the CAE tasks. As with Use-Cases #1.3 and 1.4, the conversion can be done in multiple stages, directly or indirectly through an intermediate neutral file (e.g., STL).

**Examples and Interoperability Problems.** Mesh-based analysis has turned into an industry standard for computational physics. Examples of mesh-based CAD-CAE integration are abundant in both commercial PLM systems (e.g., SolidWorks integrated with COMSOL MultiPhysics) and open-source libraries that provide APIs to various CAD systems (e.g., ENGYS).

Traditional FEA solvers require the geometric models to be tessellated into high-quality polyhedral mesh. Both tetrahedral and hexahedral mesh generation are extensively studied disciplines, and so are mesh-based analysis methods (e.g., FEM and CFD), their various extensions (e.g., X-FEM, G-FEM, and others), and their convergence and sensitivity characteristics. High quality tetrahedral meshing is easier to obtain for complex solid models, whereas hexahedral meshing remains challenging in general, though it is considered more desirable for higher-fidelity physical analysis with fewer finite elements. Both methods are known to have failure modes in complex practical scenarios without strong theoretical guarantees. One practical solution is to remove small features that can complicate meshing, though are perceived to have little impact on analysis, manually or automatically. Another approach to alleviate meshing failures is to inspect a flawed mesh for topological problems (e.g., non-manifoldness), geometric problems (e.g., slivers), and other issues, and attempting to heal them automatically. Once the CAD model is converted to a good quality mesh, analysis operations (Tasks 1, 2, and 3) are straightforward and their properties are well-understood on the polyhedral representation, as discussed in Use-Case #2.2.

Voxelization is another, arguably simpler solution based on representation conversion. Although unpredictable failure modes observed in conforming mesh generation are eliminated in non-conforming voxel maps, it comes at a cost of losing important topological properties (e.g., small voids) or geometric properties (e.g., surface normal/curvature) that can impact physical simulation. It is not often clear what the proper voxel resulting is, to avoid these problems without incurring excessive time/storage costs. Voxel maps require further assumptions for assigning the boundary conditions and body effects to voxel elements after conversion (Task 1 (a, b)), enforcing boundary conditions on non-conforming voxels (Task 2), and integrating over the implicit boundary cutting through the voxels (Task 2 (b)), whereas volume integrals turn into simple discrete summations (Task 2 (a)). A additional advantage of voxel maps is prefect parallelization (e.g., on GPUs).

By converting the representation to one that can gracefully handle analysis operations (Tasks 1, 2, and 3), the CAD-CAE integration problem is effectively turned into one of geometric interoperability, studied in the context of CAD-CAD data exchange (Use-Case Schema #1). Thus the first step to ensure property-based interchangeability of the geometric representations, which were extensively studied in Use-Cases #1.3 and 1.4 based on data-centric translation.

The challenge to address specifically in this use-case scenario is to identify which shape properties are proper surrogates for physical analysis (Tasks 1, 2, and 3). We are not aware of a rigorous theory to support this, and practical solutions appeal to arbitrary assumptions. For example, converting to polyhedral meshes or voxel maps is based on the implicit assumption that higher-order differential

properties (e.g., surface curvature) that are lost in translation do not matter. High-resolution voxelization relies on the implicit assumption that small topological features (e.g., internal voids) that are not captured by uniform sampling can be omitted. The soundness of these assumptions depends on the physical problem at hand; for instance, while small features may not impact compliance and deformations in a structural analysis problem, they will certainly change the strength and fatigue life of a model due to stress concentrations. Similarly, surface curvatures are more important in dynamic problems that deal with mechanical contact (e.g., cams and followers).

To make the matters more complex, mesh simplification and de-featuring may violate shape interchangeability in the hopes of minimally affecting physical simulation, i.e., behavioral interchangeability. But the heuristics used in manual or automatic de-featuring are not always (though should be) driven by examining their impacts on simulation (Tasks 1, 2, and 3). Similarly, mesh repair and 'beautification' algorithms should consider the ultimate impacts when modifying the overall shape or individual elements. Some of these effects are understood qualitatively in the context of mesh 'quality'—e.g., small aspect ratios tend to result in better analysis results. Below, we formalize these in the language of interoperability:

The correspondence problem is to develop conforming or non-conforming mesh generation algorithms, including voxelization as a special case with repeating elements, such that shape properties that can impact the analysis operations (Tasks 1, 2, and 3) are preserved. The extent to which the generated mesh conforms to the boundary of the original (i.e., master) CAD model can be formalized in terms of interchangeability with respect to shape (geometry and topology) properties. For instance, conforming tetrahedral or hexahedral meshes can (in principle) approximate the same solid with coarser resolution and fewer elements than voxel maps. The approximation is qualified by the invariance of geometric and topological properties of the mesh compared against the CAD model in terms of Hausdorff distance, homology, manifoldness, or any other set of pre-specified properties.

The characterization problem is to examine a given mesh generation algorithm, possibly followed by de-featuring and clean-up if necessary, and identify which shape properties are preserved along with their implications for physical/behavioral properties. Since it is difficult to theoretically reason about these properties, one solution is to investigate them on representative benchmark problems for which near-exact solutions are either known theoretically or computed by alternative methods. In a sense, this require making informed speculations on the significant properties, forming hypotheses, and testing them iteratively in a verification setting by trial-and-error.

---

### 4.2.3 Use-Case #2.6: Query-Based Integration (Standardizing on Functional Queries)

---

**Basic Idea/Setup.** The idea is similar to that of Use-Case #1.6. As before, the CAD system (i.e., A) and the CAE system (i.e., B) use dramatically different representations (i.e., $\mathsf{M}_\mathsf{A} \neq \mathsf{M}_\mathsf{B}$) and algorithms (i.e., $\mathsf{F}_\mathsf{A} \neq \mathsf{F}_\mathsf{B}$). However, in this case B tries to minimize its explicit representation of geometric information and, to refer to the original CAD model when such information are needed for physical simulation.

What makes query-based CAD-CAE integration attractive is how it delegates the geometric computations of analysis operations (Tasks 1, 2, and 3) back to where they belong i.e., on the solid model, by the solid modeler. Thus instead of translating the model to a new approximate representation scheme (e.g., a de-featured mesh or voxel map of fixed granularity) and trying to preserve geometric properties in this lossy translation, the geometric information are queried on-demand to an adequate granularity.

**Examples and Interoperability Problems.** Although commercial strength implementations of this kind (e.g., [21, 20]) are not as widespread as those of Use-Cases #2.2 and 2.4, many existing meshless/meshfree analysis methods (e.g., immersed boundary methods [53, 46, 50], radial-basis point cloud methods [47, 38], etc.) can in principle be encapsulated in a functional form to emulate a query-based integration scenario.

The defining characteristic of a query-based approach is that every geometric computation for the purpose of physical analysis in B is supported by a finite collection of queries asked from A (sequentially or in parallel). Ideally, B need not have an exploit global representation of geometry and should query everything in a local and lazy fashion.

In query-based analysis, the basis functions have inevitably non-conforming supports due to the separation of concerns. One way to enforce Dirichlet boundary conditions (Task 2) is to modify the basis functions by 'enrichment' functions; for instance, the distance function or a smoothed distance-like field whose isolevel set conforms to the geometric model. Since this information is not available explicitly, B obtains it implicitly by querying A for the distance to the boundary at a finite number of points. On the other hand, volume integral computations (Task 3 (a)) over partially overlapped domains can be performed by multiplying the integrands with the domain's indicator (i.e., characteristic or Heaviside) function. This implicit adaptation to the geometric model is supported by membership (i.e., inclusion: inside/outside) queries against A. Different sampling policies can be used; for instance, B can sample the query points B and bounce them against A viewed as a 'black box' that responds to batch membership queries, or it can request A to carefully select quadrature points for better accuracy. Similar compositions of queries are possible for surface integral computations (Task 3 (b)),[29] needed to enforce Neumann boundary conditions.

There are more elaborate alternative methods for both integration and enrichment of basis functions, some of which are more popular and effective than the simple ones illustrated here. The point of this discussion is that query-based approaches implicitly reference the geometry by local queries, without ever producing a complete explicit picture. The minimal requirements are membership and distance queries, while other queries can help achieve better performance or accuracy.

Most physical analysis problems can be composed from a finite number of membership and distance queries, which are supported by most CAD systems. The CAE system can thus delegate fundamental geometric computations to the CAD system, while focusing its attention and specialization on performing task-specific computations on the answers to those queries—in this case integration, setting up and solving systems of linear equations, etc. In general, the key to the success of query-based approaches is to formulate the interfaces in terms of queries that are consistently interpreted by both systems with respect to a common external reference. It is best to keep the interface as small as possible, and queries as simple as possible, to facilitate agreements on semantics.[30]

In a query-based approach, the interoperability map is implicit. The correspondence problem is solved by setting up the queries and restating the standard analysis operations (Tasks 1, 2, and 3) purely in terms of a composition of responses to a finite collection of queries. The surrogate shape properties are automatically preserved (by design) because the property function is computed by the same party that contains the geometric data (i.e., the CAD system).[31] As such, the approach provides a priori verification for free, only with respect to surrogate shape properties, but requires a posterior verification with respect to behavioral aspects.

[29] A practical compromise is to use a surface mesh to specify boundary conditions (Task 1 (a)) and discretize surface integrals over the polygonal (e.g., triangular or quadrilateral) elements (Task 3 (b)). 2D meshing is not as much of a bottleneck as 3D meshing, so its substitution with pointwise queries is less advantageous. This is an example of a combined data-centric and query-based method to achieve the best of both worlds.

[30] Otherwise, every data exchange can be viewed (in principle) as a large query, blurring the boundary between data-centric and query-based approaches.

[31] Mathematically, unlike the other use-cases in which a posterior verification amounts to checking if $\mu_{\mathsf{B}} = (\mu_{\mathsf{A}} \circ \eta_{\mathsf{AB}})$ (i.e., whether the triangular diagram in Fig. 17 commutes), in this case the CAE system's geometric property functions are *defined* as the composition $\mu_{\mathsf{B}} := (\mu_{\mathsf{A}} \circ \eta_{\mathsf{AB}})$ in the first place.

**Summary of CAD-CAE Integration Scenarios.** Solving the weak form of initial/boundary value problems reduces to a linear system of equations that is assembled from a knowledge of initial/boundary conditions and integral computations over specified subsets of a geometric domain. As such, the CAD-CAE integration relies on interoperability of shapes and fields supported by them (describing physical behavior), whose models are unified by implicit description of shapes as fields such as indicator or distance functions.[32] However, model-based interchangeability is even more challenging in this case than that of shapes alone. Property-based interchangeability, on the other hand, is achievable by focusing on properties that matter to predict physical behavior.

Most existing computational physics solvers are broadly categorized into mesh-based and meshfree methods, which highlights the fundamental difference between how they address these interoperability tasks. Polyhedral mesh generation is among the common approaches to constructing approximately conforming 3D (tetrahedral or hexahedral) cells to support the basis functions. The faces of the cells that conform to the boundary provide a straightforward means to assign and enforce boundary conditions (Tasks 1 (a) and 2). The volume and surface integrations (Tasks 3 (a, b)) are also reduced to those of polyhedral and polygonal domains, fully overlapped with the geometric model's interior and boundary, for which quadrature rules are well-established. However, accurate and high-quality meshing is difficult to automate. Most meshing algorithms come with no guarantees, and industrial workflows often depend on costly manual mesh simplification (e.g., de-featuring) and/or arbitrary heuristics for mesh healing/repair solutions. On the other hand, mesh-free methods circumnavigate these challenges by decoupling the geometric model from the analysis basis. However, they have to deal with locally incorporating the geometric information to quantify its effects on the integrals computed over the intersection regions within partially overlapped supports. Therefore, while mesh-based analysis fits better into a data-centric setting (i.e., geometric model passed "over the wall"), meshfree analysis lends itself better to a query-based setting with interactive and local streaming of geometric information on-demand.

[32] Every shape (i.e., set of points in 3D) can be implicitly described by a sub-/super-level set of a real-valued scalar field. The most useful such fields are the binary indicator function (useful for integration) and signed distance function (useful for enrichment).

## 4.3 CAD-CAM Integration Properties

For the purpose of this discussion, we assume that the CAM system's task is to convert the geometric specifications modeled in the CAD system to an ordered sequence of manufacturing actions. Generally speaking, the CAM analysis

- accepts the CAD model, which usually includes descriptions of shape (geometry and topology), material properties, and assembly relationships;
- accepts the dimensioning and tolerancing (e.g., GD&T) specifications, typically annotated on the CAD model's interfaces for fit/assembly; and
- computes a process plan for a set of AM/SM actions that lead to the fabrication of the part's nominal shape, respecting the tolerance specs.

We restrict our attention to additive and subtractive (AM/SM) process planning, in which the tasks are usually achieved as follows:

1. Assume that the material deposition or removal actions are modeled by sweeping a process-specific class of 3D shapes (e.g., nozzle/tool profile) along a process-specific class of rigid motions that characterize the machine's degrees of freedom (DOF). The swept region characterizes the interior (for AM) or exterior (for SM) of the action's geometric effects.
2. The manufacturing processes are broken down into manageable pieces that represent discrete manufacturing actions in such a way that there exists one or more process plans, defined as ordered sequences of those actions, that would sweep the entire internal (for AM) or external (for SM) space of the nominal geometric model and satisfy its tolerance specs.
3. If multiple plans exist, choose a subset of them that minimizes some cost function; for instance, in terms of machine hour-rates, tooling cost, material cost, and probability of failure.

Thus, in addition to the usual CAD tasks, a CAM system must support the following tasks. Each task's successful completion require preserving certain properties, discussed below:

- **Task 1 (a):** Associate prescribed dimensioning and tolerancing specs to portions of the CAD model's boundary, along with other constraints pertaining to surface quality.
  - **Property**: (Portions of) the CAD model's boundary and attributes prescribed over them.
  - **Property-based Interchangeability:** Subsets of the boundary (e.g., surface patches, and in some cases curve segments and points) that are decorated in the CAD model with annotations pertaining to manufacturing qualification must match with specifications of the process planning in terms of dimensioning and tolerancing constraints to be enforced.
- **Task 1 (b):** Associate prescribed internal/external regions of the CAD model with material properties and spatial process parameters pertaining to material deposition/removal rates, temperature, and so forth.
  - **Property**: (Portions of) the CAD model's internal or external regions and attributes prescribed over them.
  - **Property-based Interchangeability:** Subsets of the interior/exterior (e.g., volumetric cells, and in some cases lower-dimensional elements) that are decorated in the CAD model with annotations pertaining to manufacturing qualifications must match with specifications of the process planning in terms of material properties (e.g., gradation, hardness, and so on) and process parameters (e.g., deposition rate, laser power, and so on).

The assignment of these specifications to subsets of the boundary or interior/exterior need not depend on any objective or general-purpose notion of independent manufacturing features, though feature-based decomposition appear to be the most intuitive. With the current state of technology, surface specifications such as tolerance and roughness datums are usually met by finishing processes in machining, while volumetric properties such as material gradation usually pertain to additive processes in printing, but they do not have to be.

- **Task 2:** Enforce manufacturing specifications, including nominal geometry of the CAD model, dimensioning and tolerancing, and material properties.
  - **Property**: Manufacturability with a given set of manufacturing capabilities.
  - **Property-based Interchangeability:** There must exist one or more sequence(s) of manufacturing actions that produce the target part's nominal shape and pass its GD&T specifications. Each action is emulated by sweeping portions of the interior (for AM) or exterior (for SM) of the target part by the shapes defined by available machine instruments (e.g., nozzle/tool profiles) along motions restricted by machine DOFs.
- **Task 3 (a):** Find one or more (or enumerate all) process plans, i.e., sequences of manufacturing actions whose execution leads to a part that satisfies the specifications.
  - **Property**: Geometric and dimensioning specifications and material properties.
  - **Property-based Interchangeability:** The simulated outcome of execution of the sequence of manufacturing actions in the planned order, modeled procedurally in terms of set operations (e.g., union for AM and subtraction/intersection for SM) applied to the swept regions per manufacturing action must satisfy the specifications.
- **Task 3 (b):** Find one or more near-optimal process plans that minimize a cost function, subject to the above constraints (single or multi-objective optimization).
  - **Property**: Feasibility and optimality of the process plans.
  - **Property-based Interchangeability:** The cost of manufacturing, which depends on the sequence of actions, should be minimized while satisfying the design specifications and manufacturing constraints. The cost function is usually proportional to the volume of deposited or removed material in each action, where the proportionality coefficients for each action depend on machine hour-rates, material deposition or removal rates, tool utilization and wear, price of materials, and so forth.

Different feasible process plans are often found, leading to the same nominal geometry and other design specifications (e.g., GD&T and material properties). Their manufacturing cost varies; for instance, a process plan that uses combined AM+SM actions may potentially be offer more efficient fabrication than a pure AM or pure SM sequence. At a lower level, a machining process plan that removes larger volumes using a heavy-duty tool (i.e., 'roughing') to obtain a near-net shape before shaving it down to obtain the desired tolerances (i.e., 'finishing') is much more time-/cost-effective than a process plan that uses a single tool for both. All of these process plans may be interchangeable with respect to the design specifications, but not with respect to cost requirements.

- **Task 4:** Perform low-level process planning, such as tool path generation within each manufacturing action its and translation to machine instructions.
  - **Property**: Realizability (at least in the limit) of the near-optimal process plan
  - **Property-based Interchangeability:** There exists one or more low-level realizations of the high-level plans, with sufficiently high-resolution discretization (e.g., to G-code), sufficiently dense space-filling tool path generation, and so forth.

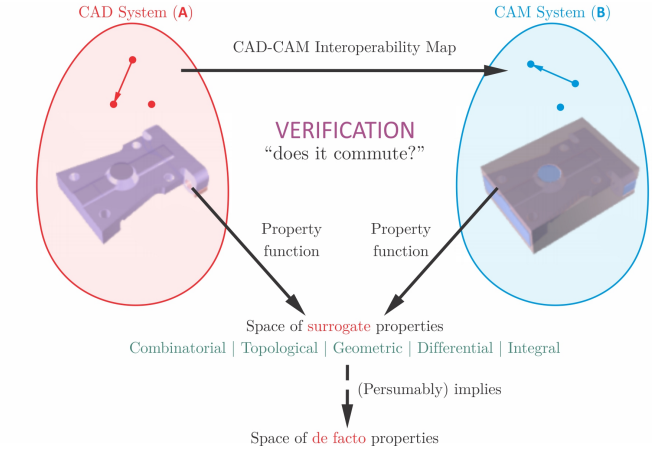



Figure 26: CAD-CAM interoperability verified up to invariance of surrogate properties.

To be clear, CAD-CAM integration may involve other tasks, such as in situ simulation of melting and mixing for 3D printing or chip formation for machining, but we will limit our discussion in this document to interoperability with respect to the above tasks and properties only.

Depending on the mechanism for CAD-CAM interoperability and each system's representation schemes, these tasks may be challenging to varying degrees. For example, Task 1 (a) is trivial in any CAM system that can accesses GD&T datums on the solid model's boundaries, assuming that the CAD system internally supports dimensioning and tolerancing. Task 3 (a) becomes easier if the CAD semantics (e.g., feature-trees) provide explicit information about independent features (e.g., holes, pockets, slots, and so on) with specialized tooling. Task 4 requires generating low-level machine code per action, which is easier if the actions are based on noninteracting features extracted from the CAD system. Tasks 2 and 3 are more challenging in general.

## 4.4 Use-Case Schema #3: CAD-CAM Integration

By analogy to the shape data exchange scenarios in Section 3 (Use-Case Schema #1), here we present 3 out of 6 conceivable scenarios for CAD-CAM integration:

1. **Use-Case #3.2:** standardizes on a common basis to represent both geometric shapes and manufacturing motions. It is the most obvious way to preserve shape properties. However, since process planning requires supporting complex shape and motion interactions, the representations that provide sufficient utility for those process-specific computations are fairly limited for fine-tuned CAD modeling.

2. **Use-Case #2.5:** standardizes on a symbolic language (feature-based or otherwise) that maps to geometric models as well as manufacturing processes. For scenarios in which the sequence of manufacturing actions can be inferred from the CAD model semantics and design intent, procedural integration provides a straightforward solution. When such semantics are not available, the manufacturing actions can be identified by feature recognition, whose drawback is the lack of universal semantics. Alternatively, the systems can converse using the language of Boolean logic at a bare minimum.

3. **Use-Case #3.6:** relies on complete delegation of geometric computations required by the CAM system to the CAD system. The global geometric information are never explicitly represented in the CAM system—or minimized to a bare minimum. Rather, the information is queried locally and lazily when needed by the aforementioned analysis tasks.

### 4.4.1 Use-Case #3.2: Standardizing on Representations (with Different Algorithms)

**Basic Idea/Setup.** The setup is similar to those of Use-Cases #1.1 or 1.2. The CAD system (i.e., A) and the CAM system (i.e., B) use the same representations (i.e., $\mathsf{M}_\mathsf{A} = \mathsf{M}_\mathsf{B}$), or at least the same core data structures, but differ in the algorithms (i.e., $\mathsf{F}_\mathsf{A} \neq \mathsf{F}_\mathsf{B}$)—namely, $\mathsf{F}_\mathsf{A}$ is a collection of algorithms that support geometric modeling operations such as Boolean operations, free-form surface editions, and so on, while $\mathsf{F}_\mathsf{B}$ is a collection of algorithms to support manufacturing planning, including accessibility analysis, tool-path planning, and so on.

**Examples and Interoperability Problems.** A number of commercial strength PLM packages come with integrated CAD-CAM solutions for machining (e.g., turning and milling), 3D printing, EDM wire-cut, sheet metal stamping, and other processes. Some offer turnkey CAD-CAM solutions in a single environment, while others offer plug-ins (e.g., Mastercam for SolidWorks). Many of these solutions directly use the CAD representation (e.g., B-reps) to generate low-level machining actions such as offset tool path generation (Task 4). However, high-level spatial planning (Tasks 2 and 3) often relies on user input to identify individual features (e.g., pockets) or make cost-effective choices pertaining to fixturing and toolset.

Volumetric enumerations such as voxels are popular as a common basis for spatial planning. The main reason is the ease with which configuration space (i.e., $\mathsf{C}-$space) computations and morphological operations such as Minkowski operations, dilation/erosion (i.e., sweep/unsweep), which are fundamental to spatial reasoning for simulating both AM/SM actions (Task 2), can be implemented on voxels using methods from image and signal processing. Further, Boolean expressions that simulate sequences of these basic actions (e.g., union for AM and subtraction/intersection for SM) (Task 3) reduce to fast bitwise operations on voxels. Both classes of operations can be computed rapidly by massive parallelization (e.g., on GPUs). However, voxel representations of swept volumes poorly approximate smooth surfaces resulting from contour-machining actions. A bigger issue is that axis-aligned voxelizations are not closed under rigid rotations. Although coordinate axis-aligned grids of voxels correspond to discrete subgroup of rigid motions for digitization of simple classes of sweeps—for instance, translational sweeps used in $3-$axis milling or revolute/helicoidal sweeps for turning—general rigid motions for high-axis multi-task CNC machines cannot be digitized.

Another example is using stacks of 2D slices to represent 3D printed parts. As with voxel maps, sliced geometry supports very limited modeling activities such as Boolean and Minkowski operations, but not free-form surface editions. Therefore, it makes a poor CAD representation, despite being ideal for integration with 3D printing software and low-level planning activities such as infill structure and support structure generation for a stack of slices or contour generation per slice (Task 4).

Since both CAD and CAM use the same geometric basis in this use-case scenario, shape-related properties remain invariant with the representation. The correspondence and characterization become trivial, and verification is guaranteed a priori. Of course, the elimination of the interoperability problem as such comes at an additional cost of case-specific challenges with making the same representation work for both CAD and CAM applications.

### 4.4.2 Use-Case #3.5: Generic Model Integration (Procedural or Declarative Recipes)

**Basic Idea/Setup.** The setup is similar to those of Use-Cases #1.5. Unlike the previous use-case, the CAD system (i.e., A) and the CAM system (i.e., B) use dramatically different representations (i.e., $\mathsf{M}_\mathsf{A} \neq \mathsf{M}_\mathsf{B}$) and algorithms (i.e., $\mathsf{F}_\mathsf{A} \neq \mathsf{F}_\mathsf{B}$).

The CAD representation is first exported to a largely symbolic procedural recipe, which can be interpreted by the CAM system for process planning. As with Use-Case #1.5, the generic model may be feature-based (e.g., generative) or constraint-based (e.g., declarative), and may include algebraic operations, global or local modifications, parametric or variational constraints, etc.

**Examples and Interoperability Problems.** CSG trees are prominent examples for this use-case,[33] as long as their primitives capture basic manufacturing actions that respect machine DOFs and minimum feature size constraints. For example, uniaxial turning is restricted to axisymmetric primitives whose external space is sweepable by the available tool inserts. 3−axis milling contributes primitives that are translational sweeps to cover the part's external space at a given fixture configuration. 3D printing contributes primitives that are translational sweeps to cover the part's internal space at a given build orientation. For a given collection of primitives, Boolean operations model discrete material deposition or removal actions (Task 3), while discrete rigid motions model re-fixturing. Importantly, Boolean logic provides the basis to decouple spatial reasoning, encapsulated by the generation of primitives by geometric modeler, from combinatorial search, which is the language of AI planning/search embedded into the process planner. The challenge is that modeling is almost never based on the same primitives that represent unit manufacturing steps, which means that the CAD representation needs to be converted to the procedural representation for CAM.

Feature-based representations are quite popular for manufacturing process planning.[34] However, feature-based methods fail frequently in the presence of complex interacting features [70]. New levels challenges are introduced due to tolerance specifications. Particularly, datum features need to be assigned correctly and tolerance stack-ups should be accounted for, using GD&T standards, which are difficult to automate for complex features.

On the other hand, there are feature-free methods that compute manufacturing primitives in terms of maximal depositable/removable regions for a fixed build/fixture configuration and a given nozzle/tool profile, without colliding with the surrounding obstacles. These regions are not tied to any specific feature semantics, and can be computed using fairly universal definitions based on mathematical morphology (e.g., opening/closing operations). The Minkowski operations from which these morphological constructions are derived can be added to the CSG semantics alongside Boolean operations, to form the basis for effective "seamless" integration of CAD and CAM systems.

This scenario assumes that the design construction in A can be exported into a procedural recipe (e.g., CSG trees) in terms of a finite number of basic manufacturing actions, and B can reproduce them for manufacturability analysis (Task 2), manufacturing planning (Task 3), and tool-path generation (Task 4). The key is to make sure that both systems agree on the representation of primitives and semantics of operations applied to them. In the case of CSG trees based on Boolean operations with possible extensions to cover Minkowski operations, diation/erosion (i.e., sweep/unsweep), and opening/closing, the common semantics are provided by mathematical morphology. The challenge remains to represent the geometric primitives consistently, such that the

[33] In fact, developing computational foundations to support CAD-CAM (particularly for NC machining back then) was the very motivation for emergence of CSG as a representation of choice for solid modeling in the 1970s [72].

[34] The mapping from the CAD model to a feature-based recipe can use methods ranging from convex volume decomposition [52, 35] and graph-based heuristics [32, 74] to rule-based pattern recognition [71, 6].

as-designed and as-manufactured models are interchangeable with respect to manufacturing specifications (e.g., GD&T).

The difficulty with feature-based methods is the lack of a common standard for feature semantics on which to base the properties for interchangeability. There is no consensus on a correct definition of 'features' across design and manufacturing literature, and most ad hoc definitions restrict one to taxonomies of narrow usability. But once the semantics are agreed-upon for manufacturing operations, the primitives can be represented by both systems in simple parametric terms and the exchange becomes largely symbolic. Feature-free methods do not suffer from arbitrariness of feature semantics, as morphological notions such as maximal depositable/removal regions are well-defined mathematically. However, interchangeable representation of primitives is more difficult in this case, because complex $\mathsf{C}-$space morphological operations lend themselves better to sampling (e.g., point clouds or voxels) than CAD-friendly representations (e.g., parametric B-reps). Below, we formalize these in the language of interoperability:

The correspondence problem is to establish a decomposition of the CAD model to atomic building blocks based on which a procedural recipe (feature-based or otherwise) can be constructed and used by the CAM system for manufacturability analysis (Task 2), manufacturing planning (Task 3), and tool-path generation (Task 4). The evaluation of the as-planned model must be interchangeable with the original as-designed model with respect to manufacturing specifications (e.g., GD&T), considering both high-level sequence of actions and low-level representations (e.g., G-code), as well as tolerance stack-ups. For feature-based methods, the interoperability map comprises of feature recognition and rule-based construction of ontologies. For feature-free methods, it relies on a combination of $\mathsf{C}-$space morphological operations.

The characterization problem is to examine a combinatorial space of procedural recipes (feature-based or otherwise), on which a manufacturing process planner operates, and identify the invariant properties. For dimensioning and tolerancing, this is to check what subset of GD&T specifications are satisfiable by one or more process plans constructed using the discrete space of atomic building blocks. While it is possible to reason about some nominal geometric properties prior to detailed planning simulation, it is harder to do so for tolerance requirements and material properties. For characterizing those properties, viable process plans may need to be tested individually. In a sense, this require making informed speculations on the significant properties, forming hypotheses, and testing them iteratively in a verification setting by trial-and-error.

---

### 4.4.3 Use-Case #3.6: Query-Based Integration (Standardizing on Functional Queries)

---

**Basic Idea/Setup.** The idea is similar to that of Use-Case #1.6. As before, the CAD system (i.e., $\mathsf{A}$) and the CAM system (i.e., $\mathsf{B}$) use dramatically different representations (i.e., $\mathsf{M}_\mathsf{A} \neq \mathsf{M}_\mathsf{B}$) and algorithms (i.e., $\mathsf{F}_\mathsf{A} \neq \mathsf{F}_\mathsf{B}$). However, in this case $\mathsf{B}$ tries to minimize its explicit representation of geometric information, and to refer to the original CAD model when such information are needed for fabrication planning.

What makes query-based CAD-CAM integration attractive is how it delegates the geometric computations of planning operations (Tasks 1, 2, and 3) back to where they belong i.e., on the solid model, by the solid modeler. Thus instead of translating the model to a new approximate representation scheme (e.g., feature-based or rule-based ontologies or voxel maps) and trying to preserve geometric properties in this lossy translation, the geometric information are queried on-demand to an adequate granularity.

**Examples and Interoperability Problems.** Although commercial strength implementations of this kind (e.g., [49, 48]) are not as widespread as those of Use-Cases #3.1 and 3.2, many existing implicit/analytic computing methods (e.g., FFT-based C−space computing [33, 16], measure-theoretic morphology [42, 43, 8], etc.) can in principle be encapsulated in a functional form to emulate a query-based integration scenario.

The defining characteristic of a query-based approach is that every geometric computation for the purpose of physical analysis in B is supported by a finite collection of queries asked from A (sequentially or in parallel). Ideally, B need not have an exploit global representation of geometry and should query everything in a local and lazy fashion.

The separation of concerns to geometric and spatial reasoning (handled by A) and logical and combinatorial reasoning (handled by B) has tremendous advantages for hybrid AM/SM process planning. The AI planning/search literature has myriad techniques to offer for manipulating finite logical expressions, converting them from one standard form to another, and find optimal paths in combinatorial state spaces by B. Geometric reasoning, on the other hand, deals with infinite (often continuum) Boolean operations (e.g., sweeps) best dealt with by A. Once the decision is made to break the manufacturing processes into discrete pieces, each encapsulating swept volumes per AM/SM action, A and B can interoperate using the common language of Boolean logic. Importantly, what makes this scenario different than the others is that B manipulates Boolean expressions symbolically, while every evaluation of union and intersection operations are delegated to A. Mathematically, Boolean union/intersection on geometric pointsets correspond to disjunction/conjunction on their indicator functions. The algorithmic implication is that B can sample the manufacturing workspace to an arbitrary resolution and evaluate Boolean formulae in terms of manufacturing primitive locally by membership queries to A.

On the other hand, the individual manufacturing primitives themselves can be evaluated in at least two different ways. It is either performed by internal algorithms in A if its geometric modeling engine supports Minkowski operations, dilation/erosion (i.e., sweep/unsweep), opening/closing, and so on. If A does not support morphological operations, B can locally evaluate them in a query-based fashion, as well. Mathematically, Minkowski operations on geometric pointsets correspond to convolutions on their indicator functions. The algorithmic implication is that B can sample the part and tool geometrise to an arbitrary resolution and evaluate convolutions by integrating over membership queries to A. For morphological operations that involve lower-dimensional motions (e.g., sweeps along tool trajectories) a similar formulation is possible in terms of distance queries.[35] Moreover, convolutions can be converted to simple pointwise multiplications in the Fourier domain as a result of the convolution theorem, implemented rapidly and in parallel (e.g., on GPUs) using fast Fourier transforms (FFT). In general, the key to the success of query-based approaches is to formulate the interfaces in terms of queries that are consistently interpreted by both systems with respect to a common external reference. It is best to keep the interface as small as possible, and queries as simple as possible, to facilitate agreements on semantics.

In a query-based approach, the interoperability map is implicit. The correspondence problem is solved by setting up the queries and restating the standard planning operations (Tasks 1, 2, and 3) purely in terms of a composition of responses to a finite collection of queries. The surrogate shape properties are automatically preserved (by design) because the property function is computed by the same party that contains the geometric data (i.e., the CAD system).[36] As such, the approach provides a priori verification for free, only with respect to surrogate shape properties, but requires a posterior verification with respect to behavioral aspects.

[35] An alternative formulation of indicator functions for lower-dimensional surfaces embedded in higher-dimensional measure spaces is given by composition of Dirac delta functions (approximately implemented by nascent delta functions, e.g., Gaussian) with distance functions [7].

[36] Mathematically, unlike the other use-cases in which a posterior verification amounts to checking if $\mu_{\mathsf{B}} = (\mu_{\mathsf{A}} \circ \eta_{\mathsf{AB}})$ (i.e., whether the triangular diagram in Fig. 17 commutes), in this case the CAE system's geometric property functions are *defined* as the composition $\mu_{\mathsf{B}} := (\mu_{\mathsf{A}} \circ \eta_{\mathsf{AB}})$ in the first place.

**Summary of CAD-CAM Integration Scenarios.** At some level of abstraction, AM/SM process planning is formulated breaks down into interactions of shapes and motions, such as applying motions to shapes, applying motions to motions, computing collision-free motions for shapes, and so forth. The CAD systems operate in the shape space, while CAM systems deal with both shapes and motions. As such, the CAD-CAM integration relies on interoperability of shapes and motions (describing manufacturing processes), whose models are unified by embedding the 3D Euclidean space into the 6D configuration space.[37] However, model-based interchangeability is even more challenging in this case than that of shapes alone. Property-based interchangeability, on the other hand, is achievable by focusing on properties that matter to obtain manufacturing plans.

The majority of the existing process planners rely on feature recognition [25, 61, 68, 24, 26] which face difficulties in the presence of complex interacting features [70]. For such models, an interoperability map that assigns manufacturing actions to individual features of the CAD model is not always well-defined. There are few feature-free alternative methods [49] that rely on principles of motion planning [37] and $\mathsf{C}-$space modeling [41]. Although $\mathsf{C}-$space modeling defines actions irrespective of a subjective feature taxonomy, it creates new interoperability problems between shapes and motions. Although $\mathsf{C}-$space modeling elegantly unifies models of shapes and motions,their representations do not easily interoperate. Explicit representations of $\mathsf{C}-$space obstacles—which are fundamental to computing the allowable motions of manufacturing instruments to avoid undesirable collisions with the surrounding components (e.g., fixtures)—turns out to computationally prohibitive. Hence, motions are typically represented by sampling, leading to discrete approximations of the swept volumes that model the manufacturing actions. Ensuring that the cumulative effect of the actions is interchangeable with the CAD model (Tasks 2 and 3 (a)) with respect to tolerance specs is a whole new challenge. Such challenges are best addressed in a query-based setup.

[37] Every shape (i.e., set of points in 3D) is in one-to-one correspondence to a motion (i.e., set of pure 3D translations), which can be embedded in the obvious way in the 6D group of general rigid motions (i.e., combinations of 3D translations and 3D rotations) [43].

## 4.5 Shape-Material Integration Scenarios

Other types of CAD-CAx integrations follow the patterns and contain ingredients similar to those we identified in CAD-CAE and CAD-CAM interoperability scenarios. In all cases, the key tasks include: identifying and prescribing application X-specific invariant properties, defining property functions and constructing interoperability maps, and finally validating the commutativity of the diagram in Fig. 18—or as we dubbed earlier, correspondence, characterization, and verification. Specific use case scenarios depend on the particular representational and algorithmic choices, but generally correspond to one of the six generic CAD-CAD exchange cases described in Section 3, due to the central role played by shape information in all such integrations. Fundamentally, each new CAx integration is characterized by a set of new invariant properties (often derived from or associated with shape properties) that are required to support application X.

We conclude this section with a brief discussion of a particularly important example of CAD-CAx integration, where X refers broadly to material modeling. Shape-material integration is taking the center stage in numerous emerging areas ranging from additive manufacturing (composites, 3D printing), to biomedical, geological, and material science applications. This section is not meant to provide a definitive treatment of shape-material integration issues; in contrast to very mature CAD-CAE and CAD-CAM integrations, this area is still in its infancy and is subject of active research. Our main goal is to illustrate how application of the property-based approach to interoperability to shape-material modeling yields useful insights into challenges and applicable interoperability scenarios.

### 4.5.1 Heterogeneous Shape-Material Modeling

The classical solid modeling assumes that the solid's interior is homogeneous and does not need to be modeled explicitly. Because of this assumption, classical solid modeling systems cannot support modeling and design of artifacts with heterogeneous material properties that are now commonly produced using additive manufacturing (3D printing, as well as composites) or other processes that enable manufacturing of functionally graded and cellular materials. Modeling and representation of such material structures necessitated development of a new area commonly dubbed "heterogeneous object modeling"[51, 36]. Conceptually, material properties are now viewed as fields (scalar, vector, or tensor) that are defined and vary over the usual solid model[28]. Such fields may be discrete, continuous, smooth, or defined in a piecewise manner. Typical computational tasks required to support heterogeneous shape-material modeling are construction and evaluation of the material fields over a given solid domain, as well as construction of solid domains based on desired material properties.

A closer examination of the tasks in heterogeneous shape-material modeling reveals that it is a special case of CAD-CAE integration where the objective of the boundary value problem is solve for a material field. Typically, material property values and derivatives are prescribed over portions of a solid (material features), and the rest of the field is constructed by a combination of constructive, interpolation, or constraint satisfaction methods subject to the prescribed boundary conditions[10]. This observation implies that heterogeneous shape-material modeling is governed by the properties, interoperability, and interchangeability issues similar to those we discussed in Section 4.1. All interoperability scenarios described in Section 4.2 are also directly applicable and most have already been published in one form or another.

### 4.5.2 Multi-Scale Shape-Material Modeling

Modeling of material structures at multiple size scales can be viewed as an interoperability problem between a collection of heterogeneous shape-material systems, with each system modeling the same physical object at a given scale[40]. As we discussed above, each (single-scale, heterogeneous) shape-material system is an instance of CAD-CAE integration. The additional challenges in multi-scale

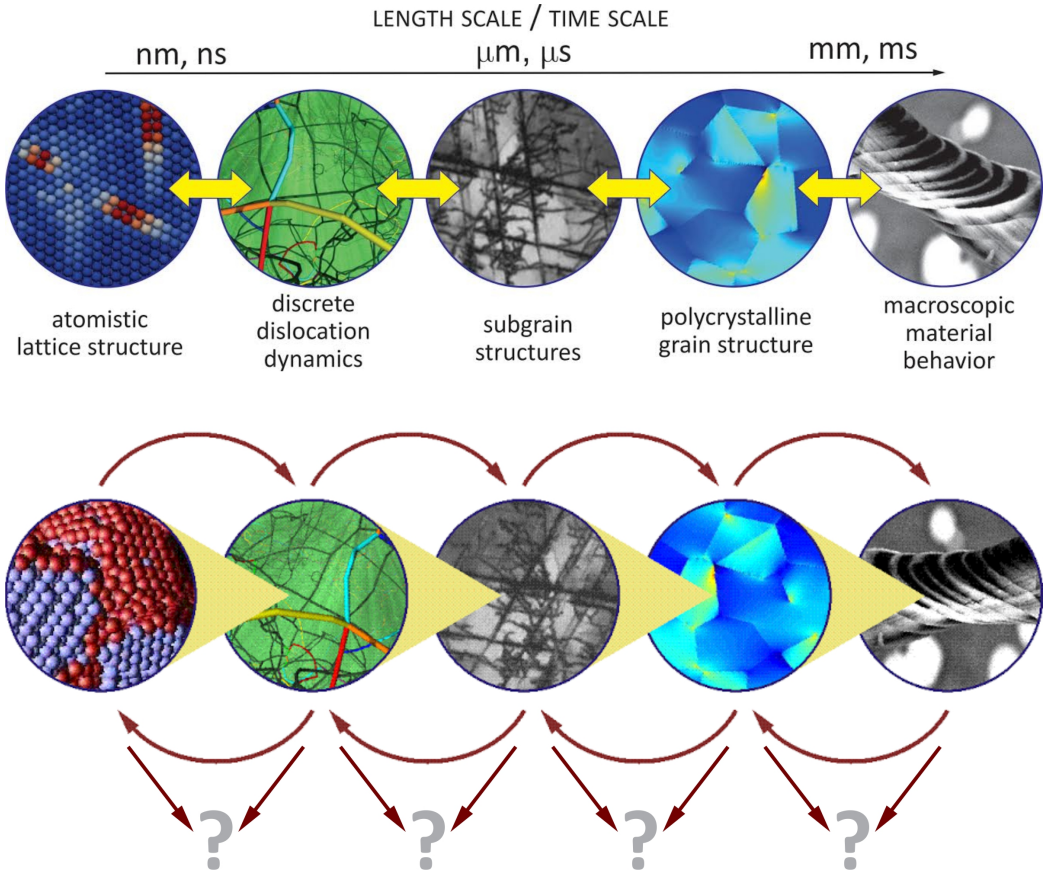



Figure 27: Multi-scale material modeling can be viewed as an interoperability problem between the scales. Figure courtesy of Dennis M. Kochmann (modified from www.kochmann.caltech.edu).

modeling arise because models, representations, and integration approaches could vary widely at each scale, but must be interchangeable because they correspond to the same physical object. For example, the same object may be viewed as a homogenized solid with spatially-varying structural and thermal properties, or as a network of parameterized unit cells, or as a collection of slices created to manufacture it using an additive manufacturing process, or polycrystaline grain structure produced by this process, and so on.

Thus maintaining interchangeable representations using different algorithms—dealing with different levels of detail—at two or more scales requires a precise specification of the shape and material invariants. Each scale comes with its own property functions, computing (surrogate or de facto) properties from its own representations and algorithms. At coarser size scales, effective material properties emerge as bulk behavior, modeled and represented via empirically or computationally homogenized constitutive relations over finite size neighborhoods, where structural randomness at lower scales is averaged out. At finer size scales, the structural details are modeled and represented explicitly. The physical behavior emerges from localized simulation at a finer level of granularity. The property functions are rich in local structural information, and governing physical laws may be different. Each scale is mapped to its higher/lower scale in the hierarchy via interoperability maps—e.g., 'homogenization' of microstructure and its nonunique inverse—that must preserve the specified properties (Fig. 27). In general, a number of difficult interoperability problems must be solved, usually in application-specific domains. These include:

- how to choose the proper sizes, locations, and shapes of neighborhoods at each size scale;
- how to formulate interchangeability of shape-material models over a collection of neighborhoods (that may include boundaries, steep property gradients, and multi-material interfaces) between any two scales;
- how to quantify the impact of the neighborhood size on the coarse-grain model for nonhomogeneous and anisotropic, and spatially varying material properties; and
- what are the proper mechanisms to systematically map the conservation, constitutive, and compatibility laws and B/I conditions from one scale to another.

All of the above challenges can be posed as either characterization problems, aiming to identify the invariants across the scales, or as correspondence problems, seeking to identify the associations between the shape-material models on neighborhoods at different scales. Tackling each problem requires extensive domain knowledge. We also note that interoperability maps across multiple scales are fundamentally different from those used in CAD-CAD interoperability scenarios in that interchangeability of models at different scales depends on non-geometric, physical, and material properties. In particular, use cases 1.1 (single system) and 1.2 (standard representation) are simply not applicable, while use cases 1.3 (System to System Translation) and 1.4 (Neutral Formats) are not particularly useful, due to rich variety of models and representations used at different scales. The remaining two scenarios: 1.5 (Generic Model Exchange) and 1.6 (Query-based exchange) are essential mechanisms for interoperability in multi-scale modeling. In particular, procedural representations are probably the most common methods for constructing the interoperability maps and solving the correspondence problem, while query-based methods are essential mechanisms for specifying and enforcing the equivalent properties (chracterization) accross multiple scales.

# 5 Conclusions

The chimera of 'model-based' interoperability results from the assumption that different representations and algorithms, authored in different systems, can be converted to each other by virtue of the identical mathematical models and functions they both intend to implement. This assumption was critical to establishing common theoretical foundations of solid modeling and creating limited (but critical) data exchange and interoperability standards in early days of computer-aided technologies. However the last several decades witnessed explosive growth in modeling techniques, representations, applications, and sophisticated systems with widely varying theoretical assumptions and computational limitations. Such systems almost never implement the exact same semantics, thus are not model-based interchangeable or interoperable. We proposed an alternative paradigm of 'property-based' interoperability, in which interchangeability is established in terms of the specific *properties* that are invariant in spite of implementation differences.

Standardize on mathematically well-defined references for properties, not file formats!

Two systems are interchangeable—only up to a certain granularity implied by the choice of properties—if they both implement equivalence classes of representations and algorithms. Each system provides property functions that map its internal elements into a common type (i.e., space of properties). The interoperability amounts to specifying a concrete mapping between the system that preserves the properties, i.e., commutes with the system-specific property functions. Obtaining this map from a specified set of properties, characterizing the invariant properties for a given map, and verifying the commutativity were respectively identified as correspondence, characterization, and

verification problems. Many unsolved computational engineering tasks and their integration into end-to-end systems can be formulated in terms of these problems.

The second important observation of this article is the following: although the properties that one deals with in design and manufacturing typically range from shape (geometry and topology) to material, physical, and behavioral properties, the shape properties typically serve as a 'surrogate' to qualify interchangeability—which is sometimes the best that one can hope for. This is justified by the fundamental role of geometry as a backbone to support other activities such as physical analysis and manufacturing planning—ranging from the topological structure of the physical laws and geometric shape of the boundary conditions in CAE to the kinematics of additive and subtractive manufacturing processes in CAM, probabilistic structural properties of materials, etc. However, in many instances, geometric interchangeability does not necessarily imply interchangeability with respect to other properties, in which case verifying the latter's invariance requires separate careful treatment.

While the focus of this report has been on interoperability of systems and applications with non-trivial geometric content, the property-based approach is broadly applicable to interoperability of *all* systems, regardless of the specific nature of such properties. For instance, recent advances in model-based systems engineering demand interoperability between variety system models described in SysML, Modelica, bond graphs, and variety of other system modeling languages. While such system models contain no geometric information, their interoperability may be formulated with respect to topological, differential, algebraic and dynamic properties[73]. Interchangeability and interoperability of such models with CAD systems or other types of CAE systems (e.g. FEA models) will require establishing invariance of other types of properties related to geometry, materials, boundary/initial conditions, integral properties, response, and so on.

# 6 Acknowledgements

This project is supported by the Defense Advanced Research Projects Agency (DARPA) under cooperative agreement HR0011-16-2-0042, and National Science Foundation grant CMMI-1547189. The content does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

The authors are thankful to Saigopal Nelaturi from PARC, Andrew Taber from Intact Solutions, Xingchen Liu from ICSI, and Randi Wang from UW-Madison, and Vaidyanathan Thiagarajan from MathWorks for the discussions that contributed to the insights in this report.

The naming of commercial products is for exemplification only and does not imply recommendation over other products. The responsibility of any omissions or errors solely lies with the authors.

# A A Recipe of Action Items for Interoperability

There are a few steps in setting up every interoperability scenario, along with a number of basic rules summarized below:

1. Identify the boundaries of your computational systems (Section 2.1):

   - What are the included subset of computational elements over which the systems are expected to have (partial) interchangeability? Specify:
     - computer representations (and their mathematical models); and/or
     - computational algorithms (and their mathematical functions);

     that should have interchangeable counterparts in both interoperating systems.

2. Determine the problem type based on knowns/unknowns (Section 2.3):

- Refer to Fig. 18: depending on whether the interoperability and/or the property functions are known, pick one of the following problem types:
- For verification problems:
  - identify the semantics of the property functions and the interoperability map (i.e., input/output types, conditions, etc.)—see the correspondence and characterization processes below for details;
  - develop theoretical foundations for proving, or experimental protocols for certifying, if the three functions compose in such a way that the interoperability map preserves all specified properties for all included computational elements, or at least, for a conclusive sample of the computational space—i.e., the output of the property functions in both systems are identical, if their inputs are mapped to each other (in either direction) via the interoperability map; and
  - [optional:] define measures of interoperability success/failure for the cases when the above condition does not hold—e.g., in terms of the frequency of its happening, discrepancies in non-invariant properties (according to some metric), etc.
- For correspondence problems:
  - define the properties with respect to which interchangeability of the computational elements will be qualified;
  - have each system provide a property function that computes a valid property—in compliance with the common semantics—for each included computational element;
  - implement an interoperability mapping (e.g., via data-centric, query-based, or a combination of the two) that preserves the invariants; and
  - having obtained both the property functions and the interoperability map, proceed to the verification process above.
- For characterization problems:
  - identify the interoperability map—i.e., a computational procedure by which the computational elements of the sending system is (or at least, can be, in principle) mapped to those of the receiving system, explicitly or implicitly;
  - identify a maximal set of properties, or as large a set as theoretically and practically feasible, which is left invariant by the mapping;
  - have each system provide a property function that computes a valid property—in compliance with the common semantics—for each included computational element;
  - having obtained both the interoperability map and the property functions, proceed to the verification process above.
- For innovation problems:
  - try to convert the problem (or part of it) to correspondence or characterization:
  - as soon as the property functions are fully or partially specified, one can proceed to the correspondence process above;
  - as soon as the interoperability map is fully or partially specified, one can proceed to the characterization process above;
  - complete the picture iteratively; and as soon as both the interoperability map and the property functions are known, proceed to the verification process above.

3. For each problem, one should keep in mind the following rules:
   - When specifying and documenting the properties and their meanings:

  - clearly specify (formally or informally) the common semantics for the properties in a language that is accessible for both systems, or their developers, to interpret correctly (automatically, semi-automatically, or manually);
  - try to leave as little assumptions, constraints, and semantics as possible open to each system's interpretation, or document the cases for which it is not possible.
  - appeal to *externally addressable* references with respect to which *measurements* (e.g., distance, size, angle, field value, etc.) can be made to define tolerance specifications for error-prone numerical properties, to ensure *transitive* interchangeability;
  - distinguish between *surrogate* properties—which commonly pertain to the geometric information backbone—and de facto properties, and document the assumptions that lead to invariance of the former indicating that of the latter; and
  - be clear and explicit about the trade-offs of some properties at the expense of others, according to which the *granularity* of interchangeability can be optimized against the limitations (e.g., intrinsic incompatibilities between the systems).
- When setting up the property functions and interoperability maps:
  - for the input data of the property functions (including parameters), use only the authoring system's computational elements and/or elements that are externally addressable by both systems;
  - for the output data of the property functions (i.e., the properties), use only the computational elements that are externally addressable by both systems (i.e., global data types and structures);
  - for the inputs and outputs of the interoperability map, chose only from the computational elements of the sending and receiving systems, respectively, in addition to global types (if needed, e.g., as input parameters); and
  - the verification simply amounts to showing that the diagram of the three functions (Fig. 17) *always* commutes (i.e., for all included representations and algorithms).

# References


[1] IEEE 610-1990. IEEE standard computer dictionary. A compilation of IEEE standard computer glossaries, Institute of Electrical and Electronics Engineers (IEEE), 1990.

[2] ISO 10303. industrial automation systems and integration – part 42: Product data representation and exchange. Technical report, International Organization for Standardization (ISO), 1998.

[3] ASME Y14.5-2009. dimensioning and tolerancing: Engineering drawing and related documentation practices. An international standard, American Society of Mechaninical Engineers (ASME), 2009.

[4] ISO 286-1:2010. geometrical product specifications (GPS) – ISO code system for tolerances on linear sizes – part 1: Basis of tolerances, deviations and fits. An international standard, International Organization for Standardization (ISO), 2010.

[5] C. Armstrong, A. Bowyer, S. Cameron, J. Corney, G. Jared, R. Martin, A. Middleditch, M. Sabin, and J. Salmon. *Djinn: A Geometric Interface for Solid Modelling*. Information Geometers, Ltd., Winchester, UK, 2000.

[6] B. Babic, N. Nesic, and Z. Miljkovic. A review of automated feature recognition with rule-based pattern recognition. *Computers in Industry*, 59(4):321–337, 2008.

[7] M. Behandish. *Analytic Methods for Geometric Modeling*. PhD thesis, University of Connecticut, 2017. Ph.D. Dissertation.

[8] M. Behandish and H. T. Ilieş. Analytic methods for geometric modeling via spherical decomposition. *Computer-Aided Design*, 70:100–115, 2016. 2015 SIAM/ACM Joint Conference on Geometric Design and Symposium on Solid and Physical Modeling (GD/SPM'2015).

[9] Rafael Bidarra. *Validity Maintenance in Semantic Feature Modeling*. Ph.D. dissertation, 1999.

[10] Arpan Biswas, Vadim Shapiro, and Igor Tsukanov. Heterogeneous material modeling with distance fields. *Computer Aided Geometric Design*, 21(3):215–242, 2004.

[11] S. B. Brunnermeier and S. A. Martin. Interoperability costs in the US automotive supply chain. *Supply Chain Management: An International Journal*, 7(2):71–82, 2002.

[12] V. Capoyleas, X. Chen, and C. M. Hoffmann. Generic naming in generative, constraint-based design. *Computer-Aided Design*, 28(1):17–26, 1996.

[13] X. Chen and C. M. Hoffmann. On editability of feature-based design. *Computer-Aided Design*, 27(12):905–914, 1995.

[14] X. Chen and C. M. Hoffmann. Towards feature attachment. *Computer-Aided Design*, 27(9):695–702, 1995.

[15] W. R. Cook. On understanding data abstraction, revisited. *ACM SIGPLAN Notices*, 44(10):557–572, 2009.

[16] B. Curto, V. Moreno, and F. J. Blanco. A general method for C-space evaluation and its application to articulated robots. *IEEE Transactions on Robotics and Automation*, 18(1):24–31, 2002.

[17] P. J. Denning. Ubiquity symposium: 'what is computation?'. *Ubiquity*, 2010, 2010.

[18] A. Edalat, A. A. Khanban, and A. Lieutier. Computability in computational geometry. In *Conference on Computability in Europe*, pages 117–127. Springer, 2005.

[19] T. Erl. *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall, 2005.

[20] M. Freytag, M. Shapiro, and I. Tsukanov. Finite element analysis in situ. *Finite Elements in Analysis and Design*, 47(9):957–972, 2011.

[21] M. K. Freytag, V. Shapiro, and I. Tsukanov. Scan and Solve: Acquiring the physics of artifacts. In *ASME 2007 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (DETC/CIE'2007)*, pages 345–356. American Society of Mechanical Engineers (ASME), 2007.

[22] Wim Gielingh. An assessment of the current state of product data technologies. *Computer-Aided Design*, 40(7):750–759, 2008.

[23] I. Gorton, G. T. Heineman, I. Crnković, H. W. Schmidt, J. A. Stafford, C. Szyperski, and K. Wallnau, editors. *Component-Based Software Engineering*. Proceedings of the 9th International Symposium on Component-Based Software Engineering (CBSE'2006). Springer, 2001.

[24] S. K. Gupta, W. C. Regli, D. Das, and D. S. Nau. Automated manufacturability analysis: A survey. *Research in Engineering Design*, 9(3):168–190, 1997.

[25] T. Gupta and B. K. Ghosh. A survey of expert systems in mnufacturing and process planning. *Computers in Industry*, 11(2):195–204, 1989.

[26] J. H. Han, M. Pratt, and W. C. Regli. Manufacturing feature recognition from solid models: A status report. *IEEE Transactions on Robotics and Automation*, 16(6):782–796, 2000.

[27] C. M. Hoffmann and R. Juan. Erep: An editable, high-level representation for geometric design and analysis. In *Selected and Expanded Papers from the IFIP TC5/WG5.2 Working Conference on Geometric Modeling for Product Realization*, pages 129–164, Amsterdam, Netherlands, 1992. North-Holland Publishing Co.

[28] C. M. Hoffmann and V. Shapiro. *Handbook of Discrete and Computational Geometry*, chapter Solid Modeling, page 57. Number 57 in Applications of Discrete and Computational Geometry. CRC Press LLC, 2016.

[29] C. M. Hoffmann, V. Shapiro, and V. Srinivasan. Geometric interoperability for resilient manufacturing. Technical Report 11-015, 2011.

[30] C. M. Hoffmann, V. Shapiro, and V. Srinivasan. Geometric interoperability via queries. *Computer-Aided Design*, 46:148–159, 2014.

[31] D. Hounshell. *From the American System to Mass Production, 1800-1932: The Development of Manufacturing Technology in the United States*. JHU Press, 4th edition edition, 1985.

[32] S.y Joshi and T. C. Chang. Graph-based heuristics for recognition of machined features from a 3D solid model. *Computer-Aided Design*, 20(2):58–66, 1988.

[33] L. E. Kavraki. Computation of configuration-space obstacles using the fast Fourier transform. *IEEE Transactions on Robotics and Automation*, 11(3):408–413, 1995.

[34] S. J. Kemmerer. *STEP: The Grand Experience*. US Department of Commerce, Technology Administration, National Institute of Standards and Technology (NIST), 1999.

[35] Y. S. Kim. Recognition of form features using convex decomposition. *Computer-Aided Design*, 24(9):461–476, 1992.

[36] XY Kou and ST Tan. Heterogeneous object modeling: A review. *Computer-Aided Design*, 39(4):284–301, 2007.

[37] J. C. Latombe. *Robot Motion Planning*, volume 124. Springer Science & Business Media, 2012.

[38] W. K. Liu, S. Jun, S. Li, J. Adee, and T. Belytschko. Reproducing kernel particle methods for structural dynamics. *International Journal for Numerical Methods in Engineering*, 38(10):1655–1679, 1995.

[39] X. Liu and V. Shapiro. Homogenization of material properties in additively manufactured structures. *Computer-Aided Design*, 78:71–82, 2016. Special Issue on the Symposium on Solid and Physical Modeling (SPM'2016).

[40] Xingchen Liu and Vadim Shapiro. Multiscale shape–material modeling by composition. *Computer-Aided Design*, 102:194–203, 2018.

[41] T. Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2):108–120, 1983.

[42] M. Lysenko, S. Nelaturi, and V. Shapiro. Group morphology with convolution algebras. In *Proceedings of the 2010 ACM Symposium on Solid and Physical Modeling (SPM'2010)*, pages 11–22, New York, NY, USA, 2010.

[43] M. Lysenko, V. Shapiro, and Nelaturi. Non-commutative morphology: Shapes, filters, and convolutions. *Computer Aided Geometric Design*, 28(8):497–522, 2011.

[44] R. Martin. Modelling inexact shapes with fuzzy sets. In *Proceedings of the Computer Science and Graphics Conference (CSG'1994)*, pages 1–26, 1994.

[45] B. Meyer. *Object-Oriented Software Construction*, volume 2 of *Prentice Hall International Series in Computer Science*. Prentice Hall New York, 1988.

[46] R. Mittal and G. Iaccarino. Immersed boundary methods. *Annual Reviews of Fluid Mechanics*, 37:239–261, 2005.

[47] B. Nayroles, G. Touzot, and P. Villon. Generalizing the finite element method: Diffuse approximation and diffuse elements. *Computational Mechanics*, 10(5):307–318, 1992.

[48] S. Nelaturi, G. Burton, C. Fritz, and T. Kurtoglu. Automatic spatial planning for machining operations. In *2015 IEEE International Conference on Automation Science and Engineering (CASE'2015)*, pages 677–682, 2015.

[49] S. Nelaturi and V. Shapiro. Representation and analysis of additively manufactured parts. *Computer-Aided Design*, 67-68:13–23, 2015.

[50] J. Parvizian, A. Düster, and E. Rank. Finite cell method: h- and p-extension for embedded domain methods in solid mechanics. *Computational Mechanics*, 41(1):121–133, 2007.

[51] Alexander Pasko, Valery Adzhiev, and Peter Comninos. *Heterogeneous objects modelling and applications: collection of papers on foundations and practice*, volume 4889. Springer, 2008.

[52] D. B. Perng, Z. Chen, and R. K. Li. Automatic 3D machining feature extraction from 3D CSG solid input. *Computer-Aided Design*, 22(5):285–295, 1990.

[53] C. S. Peskin. The immersed boundary method. *Acta Numerica*, 11:479–517, 2002.

[54] S. Raghothama and V. Shapiro. Boundary representation deformation in parametric solid modeling. *ACM Transactions on Graphics*, 17(4):259–286, 1998.

[55] S. Raghothama and V. Shapiro. Topological framework for part families. *Journal of Computing and Information Science in Engineering*, 2(4):246–255, 2002.

[56] A. A. G. Requicha. Mathematical models of rigid solid objects. Production Automation Project, Technical Memo. No. 28 (TM-28), University of Rochester, 1977.

[57] A. A. G. Requicha. Representations for rigid solids: Theory, methods, and systems. *ACM Computing Surveys*, 12(4):437–464, 1980.

[58] A. A. G. Requicha. Representations of rigid solid objects. Production Automation Project, Technical Memo. No. 29 (TM-29), University of Rochester, 1980.

[59] A. A. G. Requicha and H. B. Voelcker. Solid modeling: A historical summary and contemporary assessment. *IEEE Computer Graphics and Applications*, 2(2):9–24, 1982.

[60] D. S. Scott and C. Strachey. Toward a mathematical semantics for computer languages. Technical Monograph PRG-2, Oxford University Computing Laboratory, Programming Research Group, 1971.

[61] J. Shah, P. Sreevalsan, and A. Mathew. Survey of CAD/feature-based process planning and NC programming techniques. *Computer-Aided Engineering Journal*, 8(1):25–33, 1991.

[62] J. J. Shah and M. Mäntylä. *Parametric and Feature-Based CAD/CAM: Concepts, Techniques, and Applications*. John Wiley & Sons, 1995.

[63] V. Shapiro. *Representations of Semi-Algebraic Sets in Finite Algebras Generated by Space Decompositions*. Ph.D. dissertation, 1991.

[64] V. Shapiro. *Handbook of Computer Aided Geometric Design*, chapter Solid Modeling, pages 473–518. North Holland, 1st edition edition, 2001.

[65] V. Shapiro and D. L. Vossler. What is a parametric family of solids? In *Proceedings of the Third ACM Symposium on Solid Modeling and Applications (SMA'1995)*, pages 43–54, New York, NY, USA, 1995. ACM.

[66] B. Smith, K. Brauner, P. Kennicott, M. Liewald, and J. Wellington. Initial graphics exchange specification (IGES) version 2.0. Technical report, NTIS, Springfield, VA, USA, 1983.

[67] S. Spitz and A. Rappoport. Integrated feature-based and geometric CAD data exchange. In *Proceedings of the 9th ACM Symposium on Solid Modeling and Applications (SMA'2004)*, pages 183–190. Eurographics Association, 2004.

[68] S. Subrahmanyam and M. Wozny. An overview of automatic feature recognition techniques for computer-aided process planning. *Computers in Industry*, 26(1):1–21, 1995.

[69] R. B. Tilove. Set membership classification: A unified approach to geometric intersection problems. *IEEE Transactions on Computers*, 100(10):874–883, 1980.

[70] Y. J. Tseng and S. B. Joshi. Recognizing multiple interpretations of interacting machining features. *Computer-Aided Design*, 26(9):667–688, 1994.

[71] J. H. Vandenbrande and A. A. G. Requicha. Spatial reasoning for the automatic recognition of machinable features in solid models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(12):1269–1285, 1993.

[72] H. B. Voelcker and A. A. G. Requicha. Geometric modeling of mechanical parts and processes. *Computer*, 10(12):48–57, 1977.

[73] Randi Wang and Vadim Shapiro. Topological semantics for lumped parameter systems modeling. *arXiv preprint arXiv:1811.02666*, 2018.

[74] M. C. Wu and C. R. Lit. Analysis on machined feature recognition techniques based on B-rep. *Computer-Aided Design*, 28(8):603–616, 1996.

[75] Y. Yamaguchi, H. Nakamura, and F. Kimura. Probabilistic solid modeling: A new approach for handling uncertain shapes. In *Selected and Expanded Papers from the IFIP TC5/WG5.2 Working Conference on Geometric Modeling for Product Realization*, pages 95–108. Association for Computing Machinery (ACM), 1992.

[76] A. A. Zaldivar and J. C. Torres. Fuzzy solid models: Foundations, representations and framework. In *International Conference on Fuzzy Systems*, pages 1–8, 2010.