A CONFLICT DETECTION APPROACH FOR COLLABORATIVE MANAGEMENT OF PRODUCT INTERFACES

Onur Hisarcikililar
Department of Mechanical Engineering
McGill University
Montreal, Quebec, Canada

Keyvan Rahmani
Department of Mechanical Engineering
McGill University
Montreal, Quebec, Canada

Vince Thomson
Department of Mechanical Engineering
McGill University
Montreal, Quebec, Canada

ABSTRACT

In complex products, maintaining subsystem consistency throughout the design process is often a time-consuming process of document exchange among cooperating functions. This paper describes a conflict management approach that lead to the computer-aided management of the product specification conflicts that happen due to the integration of subsystems. In order to define a framework, a systematic interface representation which proposes building generic interface schemes for subsystem connectivity representation is described. Based on this methodology, a functional architecture of the proposed conflict management method, along with generic exception taxonomy of conflicts is developed. The applicability of the proposed concepts is discussed through illustrative examples. The proposed methodology is intended to allow automatic detection and handling of interface connectivity errors throughout collaborative design processes.

INTRODUCTION

An important part of collaborative design is managing the consistency between the subsystems of a product. However, maintaining subsystem consistency throughout the design process is often done manually through a time-consuming process of document exchange among cooperating functions. Current literature proposes some computer-aided methods to identify various conflicts between a design and its requirements [1, 2]. In concurrent engineering, a large number of conflicts can happen when integrating systems with incompatible interfaces.

So far the majority of conflict management research is devoted to that of process and design versus manufacturing; in this paper, we focus on the product specification conflicts that happen due to the integration of subsystems. Interface specifications defined for a subsystem may not fit other subsystems’ specifications interacting with it. Moreover, different groups designing different subsystems often have different perceptions regarding what might be the best design for interfaces, which may result in incompatible terminologies or sets of parameters to describe the interfaces. This inevitably results in technically inconsistent subsystem designs, which need to be addressed by conflict management methodologies.

In complex products, establishing and managing the interfaces between subsystems is very challenging. Subsystems of such products are mostly developed by different design groups, sometimes not even from the same company. This makes it necessary to clearly define subsystem boundaries and the interfaces between them in order to ensure proper functionality in the final product. Currently, none of the existing conflict management methods deals well with subsystem interfaces.

In order to define a conflict management framework, this paper considers a systematic interface representation proposed by Rahmani and Thomson [3]. This representation consists of using a generic interface scheme for subsystem connectivity representation. It is helpful for conflict detection because it allows building explicit and simple exception classification and helps to provide effective algorithms for detecting incompatibilities.
The paper first briefly explains the existing system connectivity modelling approaches and conflict detection methodologies in collaborative design. It also introduces an interface model that is suitable for the conflict management framework proposed in this paper. Next, it presents the functional architecture of the proposed conflict management method. Then, it proposes a generic exception taxonomy for the classification of conflicts. The last part of the paper discusses which methods can be used for mapping interface conflicts to the exception taxonomy.

RELATED WORK

Approaches for product system connectivity modeling

Different approaches and techniques that model connectivity information and enhance consistency between product systems definitions are proposed in the literature. This section summarizes relevant approaches for managing interfaces.

Model-based system engineering. A model-based systems engineering methodology can be characterized as the collection of related processes, methods, and tools used to support the discipline of systems engineering in a ‘model-based’ or ‘model-driven’ context [4].

In engineering design, models are used to formally represent the structure, function, and behavior of a system [5]. Additionally, experiments can be performed on models to eliminate poor design alternatives and to ensure that a preferred alternative meets stakeholder objectives. Models also facilitate designer collaboration by providing a common formalism for communicating information about the system [6].

Modeling techniques, such as SysML are used to describe inputs, outputs, sequences, and conditions for coordinating various system behavior, and to represent mathematical constraints amongst system properties. It enables the modeling of algebraic equations that establish mathematical relationships between system properties to enhance system consistency [7]. SysML can also be used, for instance, for maintaining consistency between the multiple product models and views through defining metamodels and graph transformations to map the views [8]. However, no formal ways of modeling interface definitions, as well as detection mechanisms for dealing with conflicts between subsystems are proposed.

Modeling language systems. Interface modeling techniques are also proposed within modeling language systems. For instance, Dymola [9, 10], an object-oriented system using Modelica [11] language, allows the physical modeling of large interconnected systems based on model components from different engineering domains. Models are hierarchically decomposed into submodels which are connected in accordance with physical coupling of components. Dymola allows the development of domain-specific class libraries for electronic circuits, hydraulic systems, thermodynamics systems, etc., which can be used in conjunction for generating a specific multi-domain application model [12]. Physical objects can be mapped by Dynola, and connected through equations that describe interfaces. Despite important capabilities of system and interface modeling by Dymola and Modelica, and their large spectrum of application domain (varying from vehicle powertrain design [13] to electrical storage systems design [14]), these systems are mostly intended to be used for simulation and analysis of system behavior, and do not aim to support the issue of system consistency in a collaborative development context.

Matrix-based techniques. The connectivity map for subsystems can be shown by matrix-based representations, such as the design structure matrix (DSM) [15]. DSM is a simple NxN matrix that represents relationships between entities. DSMs provide a systematic approach to recognize and analyze interfaces for the purpose of improving product architecture.

The DSM is a useful tool to analyze the structure of a product by indicating the degree of dependency between subsystems and clustering entities into modules with more or less interdependency [16]. However, despite its computational benefits, it does not provide any information about the content of each relationship. Knowing the content of each relationship is very important to keep system consistency for designing actual components.

Constraint satisfaction techniques. Constraint satisfaction techniques can model design problems through variables for representing design parameters, and constraints to express design requirements in a formal way [17].

These techniques provide the possibility to detect and manage conflicts [18] and may find potential alternatives. The formalization of project information using constraints augments the amount of information available for subsequent decisions [19].

In the context of collaborative product design, various techniques have been proposed to enhance design consistency. For instance, Bahler et al. [20] proposed a design advice tool which uses constraints to support negotiation and conflict resolution. Khedro and Genesereth [21] have developed a progressive negotiation strategy for conflict resolution where locally consistent solutions are used to converge on global consistency. However, these works are mostly focused on geometrical parameters, and deal with the conflicts between the design and its requirements, and do not directly address the issue of maintaining interface consistency between subsystems.

Techniques for conflict detection and handling

An important part of the requirements management in collaborative design is concerned with detecting and resolving conflicts between designers. In collaborative design, the conflicts are detected by a tedious and resource-intensive process. They can have a severe impact on the cost, schedule, and quality, if left unaddressed for too long [22]. Notably, in large–complex products like the ones in the aeronautics or automotive industry, the unresolved conflicts in the early stages of detailed design have huge lasting implications [2]. Conflicts in a collaborative design environment occur when at least two incompatible design commitments are made or when a design
party has a negative critique of another design party’s actions [1].

According to Lander [23], there are several ways in which conflict between design participants can be managed, such as avoidance (avoid conflict by sharing information about local constraints and priorities), conflict classification (build a taxonomy of conflict types), or negotiation (bargaining, restructuring, constraint relaxation, mediation, arbitration, etc.). This paper is focused on classification and detection of technical design conflicts in connected subsystems.

A large number of works view conflict management as an exception handling component of an integrated set of cooperating design services [1, 24, 25]. Design rationale within conflicts can be captured to help design parties identify the reasoning of design actions in order to improve the effectiveness of the conflict management process. Klein [26], for instance, proposed three underlying concepts. (1) Conflicts are hierarchically organized so that the more general conflicts, i.e., domain-independent, are at the upper part of the hierarchy, whereas the more particular conflicts, i.e., domain-dependent, are at the lower part of the hierarchy. (2) Conflict solving strategies can be organized in a hierarchy in which the more general strategies are at the upper part of the hierarchy, whereas the more particular conflict solving strategies are at the lower part of the hierarchy. (3) When a conflict occurs, it is assigned a position in the conflict hierarchy to which it belongs, and then it is mapped onto the hierarchy of conflict solving strategies in order to instantiate a conflict resolution action. A major advantage of this method is the high efficiency of detection if a conflict similar to the problem under consideration is already defined. Numerous conflict check and handling models based on the exception handling approach have been proposed in the literature (a survey is presented in [1]). None of them explicitly addresses conflict detection within interface management.

In this paper, an exception handling approach is developed for interface management methodologies. In this approach, interfaces among product subsystems are defined on the basis of a generic scheme. As the present paper demonstrates, this technique allows the building of simple yet effective exception taxonomies and algorithms for detecting incorrect or inconsistent interface definitions.

**INTERFACE MANAGEMENT**

Rahmani and Thomson [3] proposed a scheme for systematically identifying and describing product interfaces for new product development. One can begin by considering a composite interface between two subsystems which consists of the following four interface sub-classes: spatial, energy, information and material. Each of these classes can be subsequently divided into more specific types; for example, the energy interface can be divided into mechanical, electrical, and so on. The interface subtypes result in comparable interface parameters that can be used to evaluate subsystem incompatibilities.

The work done by Rahmani and Thomson [3] aims at defining the connectivity between product subsystems through a domain independent interface scheme. Interface elements are defined for each product subsystem as instances of this scheme. A terminology describing the function of the connection is associated within each interface element.

After the definition of interface elements, connectivity between subsystems is defined by interconnecting these elements via interface rules. Rules describe explicit or implicit connections between parameters of the interfacing subsystems (Figure 1).

In this figure, two subsystems are connected by a composite interface that is decomposed into its subtypes. The final primitive interface types contain a set of parameters. For example, the parameter set of a DC electrical energy include the voltage and the current. The rules try to match these parameters with the same set of parameters from the other subsystem. Hence, unmatched parameters can be reported as a conflict to the conflict management system.

![Figure 1. SUBSYSTEM CONNECTIVITY THROUGH INTERFACES](image-url)
The conflict detection model that is proposed in this section supports the detection and reporting of interface exceptions for design teams that develop interdependent subsystems. This exception handling approach allows automatic detection of correctness, completeness, connectivity and consistency of the interface definitions. The following subsection describes a functional architecture for the approach. The next section presents techniques for exception classification and detection.

**Functional architecture**

The functional architecture of the proposed system consists of the following components (Figure 3):

- a subsystem decomposition,
- an interface database,
- a product-dependent interface library, and
- an exception detection engine.

The designers add new interface elements about their subsystems into the database. The exception detection engine executes the consistency check and reports the exceptions to the designers. The interface library contains standard interface definitions related to the product, and is used as a support for the completeness and correctness check. Parts of the architecture are detailed below.

**Product subsystem decomposition.** A multi-disciplinary complex product is composed of systems from different engineering domains that need to be connected. Each of these systems are modeled and managed by a domain specific tool. Hardware-software interfaces are a good example of this type of complexity.

Hence, the proposed interface exception detection system considers a subsystem decomposition of the product. Interface definitions are used for interconnecting these subsystems. Interfaces can be defined and managed at different levels of the decomposition.

**Interface database.** Interface elements that define the connectivity between the product subsystems are stored in the interface database. The database is shared among the design stakeholders. The stakeholders contribute to the database by adding or modifying the elements with regard to their systems.

For each subsystem, a series of interface definitions describes its connections to the other subsystems. Each interface definition in the database consists of an interface element, a structure derived from the interface taxonomy and an associated terminology, along with metadata that include the subsystem where it is defined, the target subsystem towards which it defines a connection, and the target interface element within the target subsystem.

Figure 2 presents an example of an interface element. This element defines an electrical connection from a battery to an electrical engine.
This definition is subject to a comparison with the interface element on the engine side that defines the engine’s electrical energy requirements. One should expect that the requirements on the engine side are defined by an interface element with the same types of attributes along with compliant values or expressions, such that the interface definitions are considered as consistent.

**Interface template library.** Interface template library contains product-dependent standard interface definitions. These definitions are intended to be used within design projects and across different design stakeholders, e.g., an OEM and its suppliers. An interface definition library also allows the machine recognition of interface definitions, i.e., an automatic correctness and completeness check of the interface database elements on the basis of standard definitions.

Library definitions are also derived from a generic interface taxonomy; however, they define all the possible combinations of attributes and terminologies that allow a complete definition of a type of interface. Figure 4 gives an example of an electrical connection template. The template prescribes that an electrical connection must be named either as “electrical routing”, “electrical connection” or “electrical wire”, and must contain both energy and spatial attributes. The energy attribute type must be electrical, which may be either AC or DC. The AC connector may be defined in two ways, either with a combination of voltage, current and voltage drop values, or a combination of impedance, voltage and impedance drop values. Spatial attributes must define topology attributes of plug type and diameter. The plug type may have either “male” or “female” attributes.

The level of generality of the templates may differ, i.e., design teams may prefer more generic or more specific templates with regard to their convenience. For instance, one can define in the library either a unique “electrical connection” template, or two distinct “AC connection” and “DC connection” templates.

**Exception engine.** The exception engine undertakes conflict detection for interface definitions. The conformity of the database definitions with regard to both the target interfaces and the standard definition in the library are checked. Some techniques for that mechanism are presented in the following section.

**Interface conflict classification and detection**

This section presents the interface exception taxonomy along with mechanisms for conflict detection and mapping.

**Conflict classification: the exception taxonomy.** As mentioned before, the proposed conflict detection approach consists of mapping the interface exceptions within an exception taxonomy. The proposed taxonomy is subdivided into two main categories, element and relationship exceptions (Figure 5).

---

**Figure 4. EXAMPLE OF INTERFACE TEMPLATE FOR ELECTRICAL CONNECTION**

**Figure 5. MAIN EXCEPTION CATEGORIES IN THE EXCEPTION DETECTION ENGINE**
subdivided into two categories: completeness where the element contains missing information, and correctness where it contains incorrectly defined information. For each of these types, elementary exception sub-types are defined (Figure 6).

- **Completeness exceptions** occur when an interface element is missing attributes or scheme parts. For instance, an unspecified parameter value or terminology exception detects non-instantiated parameters or terminologies. An incomplete attribute set exception occurs when an interface element does not conform to the default definition in the library.

- **Correctness exceptions** when an interface element has incorrectly defined attributes or terminology. For instance, an incorrect attribute set exception occurs when the attributes of the interface element do not match standard library definitions.

**Figure 6. ELEMENT EXCEPTION TAXONOMY**

- **Completeness**
  - unspecified parameter value
  - incomplete attribute set
  - unspecified terminology
  - element duplication

- **Correctness**
  - incorrect attribute set
  - incorrect terminology

**Figure 7. RELATIONSHIP EXCEPTION TAXONOMY**

**Conflict detection: interface type recognition and exception mapping.** The interface representation methodology that is used in this approach leads to the use of simple techniques for conflict detection. A structural consistence (SC) calculation methodology is presented in this section as an example. The SC consists of comparing the branches (or elementary attributes) of interface elements. The structural consistence value between two interface elements is useful for:

- checking inconsistencies between connected interface elements,
- checking completeness and correctness of an interface definition through comparison between an interface element and a library template, and
- matching a specific interface element to the corresponding template in the library.

Structural consistence between two interface elements is determined through the difference between the total number of branch types defining the two interface elements and the number of unique branches, i.e., ones that are not common to both elements. The structural consistence value is:

$$ SC = \frac{TB - UB}{TB} $$

where TB indicates the total number of branch types and UB the number of unique branches. Note that SC equals 1 when the two elements are perfectly consistent and 0 when they are totally inconsistent.

Structural consistence between an interface element and a library template consists of considering every possible complete definition that can be derived from the template. The maximum SC value within these determines the SC between
the template and the element; that is, the structural consistency between an interface element and a template ‘a’ is:

$$SC_a = \max \left( \frac{TB_{ai} - UB_{ai}}{TB_{ai}} \right) \text{ for } \forall i$$

(2)

where ‘i’ indicates the value for the $i^{th}$ definition derived from the template $a$. Note that the interface element is considered correct and complete with regard to the template ‘a’ when $SC_a$ equals 1. The interface element is considered incomplete with regard to $SC_a$ when $UB_{ai}$ concerns only the attributes of $SC_a$ and incorrect when at least one of the $UB_{ai}$ concerns an attribute of the interface element.

To match a specific interface element to a template within the library, $SC$ values between the element and all of the templates in the library are compared. The template that gives the maximum $SC$ value is considered as the matching template. That is, the template ‘t’ matches an interface element when:

$$SC_t = \max( SC_a ) \text{ for } \forall a$$

(3)

where ‘a’ indicates the $a^{th}$ template in the library. Note that the interface element is considered as correctly and completely matching to the template library when $SC_t$ equals 1.

ILLUSTRATIVE EXAMPLES

The following examples illustrate how the exception handling taxonomy proposed in this paper can be useful in a design process.

Pneumatic actuator

Figure 8 shows a system composed of a logic controller, a set of pneumatic valves and some actuators that are actuated by the valves. It is assumed that in this example the valves subsystem, the controller and the actuators are designed collaboratively by different groups.

The air flow rate and pressure are two indispensable attributes for representing the connectivity requirements of the actuators. Within the interface library, the default pneumatic connections are represented along with these two parameters. An unspecified parameter value exception occurs when the air flow rate and pressure are undefined when defining the pneumatic connection specifications of the valves. It is quite useful as well from the system design point of view, if the output ports of the controller and the electrical connections of the valves are named consistently; otherwise, a mismatching terminology exception occurs.

The voltages of the controller output ports and the valve electrical connections should match. Consequently, a violated rule exception occurs if the valves require 24V, but the controller interface only provides 5V.

CNC machining station

Figure 9 shows another example, where a special CNC machining station is to be designed by different engineering teams. The machine bed is designed concurrently with the automatic fixture that holds the work piece in position. A separate hydraulic unit is also designed to provide the fixture and some other machine parts with hydraulic pressure. The hydraulic system is only connected to the machine bed from which the hydraulic oil is transferred to different machine parts as well as the fixture. The fixture is positioned and fixed on the machine bed by means of two locating pins, and a set of bolts (not shown).

In this design case, the pins are important interface components. The system designers should decide to which subsystem the locating pins belong. If both parties design the pins, an element duplication exception occurs.

On the fixture, the tolerances of the pin positions corresponding to the holes in the machine bed have to be defined in the same way to avoid integration problems. Otherwise, a mismatching attribute set exception occurs. A violated rule exception can also occur here if the diameters of the pins and the holes do not match.
CONCLUSION

This paper presented a conflict management approach for managing the consistency of interface definition between subsystems of a product in collaborative design. In order to define a conflict management framework, a systematic interface representation which proposes building generic interface schemes for subsystem connectivity representation was described. Based on this methodology, a functional architecture of the proposed conflict management method, along with a generic exception taxonomy of conflicts was developed. The applicability of the proposed concepts was discussed through illustrative examples.

The work presented in this paper offers insights about how product interfaces can be managed through a conflict management approach. As mentioned in the section on related work, several approaches for system interface design and consistency maintenance in engineering design are already proposed. The originality of the work presented in the present paper is the exception-based conflict detection mechanism based on a systematic approach for interface definition intended to support design collaboration. The proposed approach is developed to allow different design groups defining the interfaces for their respective subsystems independently, and checking the correctness and consistency of these definitions in an asynchronous way. The proposed conflict management techniques offer two main benefits: (1) accurate detection and type recognition of exceptions, and (2) simple detection algorithms for elementary exception types.

Especially for complex products composed of thousands of interacting components, e.g., an aircraft, changes occur frequently throughout the product lifecycle and the large number of interconnected components cause many difficulties in common attributes with regard to their respective impacts on change.

ACKNOWLEDGEMENT

The research reported in this paper was undertaken as part of the “Collaborative Development for Product Life Cycle Management” (PLM2 CRIAQ) project. The authors are grateful for the financial support from Natural Sciences and Engineering Research Council of Canada (NSERC), Consortium for Research and Innovation in Aerospace in Québec (CRIAQ), Pratt & Whitney Canada Corp., Bombardier Inc., CAE Inc., CMC Electronics Inc., and Rolls-Royce Canada Limited.

REFERENCES


