Contents lists available at ScienceDirect

Automation in Construction

journal homepage: www.elsevier.com/locate/autcon

Multidisciplinary Design Optimization through process integration in the AEC industry: Strategies and challenges

Héctor Díaz [a,*], Luis F. Alarcón [a], Claudio Mourgues [a], Salvador García [b]

[a] School of Civil Engineering, Pontificia Universidad Católica de Chile, Vicuña Mackenna 4860, Macul, Santiago, Chile
[b] School of Civil Engineering, Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Monterrey, Eugenio Garza Sada 2501 Sur, Monterrey, Mexico

ARTICLE INFO

Article history:
Received 28 July 2015
Received in revised form 31 August 2016
Accepted 20 September 2016
Available online 8 November 2016

Keywords:
Building Information Modeling
Process Integration and Design Optimization
Multidisciplinary Design Optimization
Parametric Modeling
Component Interoperability
Architecture-Engineering-Construction
MDO requisites

ABSTRACT

Recently Multidisciplinary Design Optimization (MDO) has emerged in the Architecture-Engineering-Construction (AEC) industry to assist designers in making the design process more efficient, by achieving more design alternatives in less time. Currently, MDO is developed with software tools that work together and automatically. However, the technical requisites to develop MDO using Process Integration and Design Optimization (PIDO) platforms are not clearly specified in the design optimization literature. There are many difficulties not covered by the literature: especially the tools' behavior, and the strategies to deal with PIDO. To determine the technical requirements, the tools' behavior, the challenges of interoperability and viable strategies, we reviewed the literature and tested five tools. This paper presents the main behavior of the tools we studied, and explains the challenges and strategies to develop MDO through PIDO. We observed three technical tool requisites: component interoperability, tool automation, and model parameterization capabilities. We detected low openness levels of the tool interfaces that did not always enable a full integration with PIDO or permit access to model properties. The scarcity of commands and the presence of pop-up menus impeded performing analyses automatically. Moreover, most of the tools did not allow parametric associations among components, compatibility among themselves or the addition of custom components. The strategies proposed focused on testing the tool interfaces, to validate that each computational process runs automatically, and to confirm that parametric relationships and components are possible. The tools tested were not specifically designed to include full capability to work with PIDO, therefore, enhancements would be needed to meet the three requisites: component interoperability, automation and parameterization. Technological, documentation and programming challenges also emerged when working with tools. We demonstrated that only certain tools can be used with a PIDO platform. However, there may be still other requisites for MDO using different methods that can become the focus of future work.

1. Introduction

The design of a building is a creative and complex process that involves multiple conflicting criteria. In addition, for traditional methods of design optimization, the iterative and generative nature of design can prove quite time-consuming [1]. These methods require tedious manual iterations that are wasteful and should be avoided or minimized [2]. Thus, the number of options and the exploration of the design domain are limited because of the time and resources consumed in these iterative cycles. This is why computing methods can provide a partial solution for such a complex design process, specifically to be used in activities related to the generation and assessment of design options.

One of the computing approaches to ameliorate this problem is the Multidisciplinary Design Optimization (MDO) method [3–6]. MDO has been successfully used for several years by the aerospace and automotive industries [7,8]. Multidisciplinary optimization is, however, a more recent approach in the Architecture-Engineering-Construction (AEC) industry and therefore, it cannot be appropriately used in most stages of building design. After the initial creative part of the design process, however, MDO can be applied to accelerate the search and assessment of design options, two processes that would be very difficult to achieve by human efforts. MDO starts from a model previously resolved by designers and an optimization problem clearly defined.

MDO is a formal methodology for the design of complex systems that explores interaction of various disciplines. It is an optimization method that addresses the design problem by decomposing it into smaller multidisciplinary parts. MDO is based on a decomposition principle that divides the main problem into several sub-problems. The design problem is then tackled by analyzing the inter-relationship,

* Corresponding author.
E-mail address: hudiaz@uc.cl (H. Díaz).

http://dx.doi.org/10.1016/j.autcon.2016.09.007
0926-5805/© 2016 Published by Elsevier B.V.

represented by variables, among the different sub-problems [4]. MDO utilizes optimization algorithms and multidisciplinary analyses to explore the design domain by modifying a virtual model parametrically. The aim of MDO is to find the optimal or near-optimal design that satisfies the multidisciplinary objectives and meets the given constraints.

1.1. Main tools to perform MDO in the AEC industry

MDO is accomplished by several types of software tools. Some of these types are optimization platforms, such as Process Integration and Design Optimization (PIDO); modeling tools, such as Computer-Aided Design (CAD); Building Information Modeling (BIM); Parametric Modeling (PM) tools and analysis tools, such as structural analysis, energy simulation, cost analysis, and so on. This section describes these tools.

PIDO is a software family that has been used in other domains, such as the manufacturing and aerospace industries. Recently, it has also been used by some AEC researchers aiming to achieve MDO. PIDO helps to facilitate and automate the integration and interoperability of software applications involved in simulation tasks. PIDO utilizes simulation workflows [9] in order to optimize one or more aspects of a product design. It is also a tool for the process modeling environment (PME) [10] because it can integrate components to create a unique process model.

CAD is the use of computer systems and software to create, modify, and analyze designs. CAD implements digital graphics in the system along with an application to facilitate projects [11]. This type of software works with both 2D and 3D representations and is broadly used in the engineering and architecture domain. In the AEC industry, CAD is used to create blueprints and 3D visualizations of a project (e.g., a building).

BIM is a Parametric Modeling technology which includes a set of processes to produce, communicate, analyze and manage building models [12]. BIM contains building information and parametric links as part of a 3D model that could be contained in a database, a spreadsheet, a text file or a schedule [13]. The stored information covers the whole life cycle of the building. The parametric rules are embedded in elements. These rules deliver intelligence to the model because they reconfigure it and allow for modifying elements and updating the model views automatically when a change occurs. The BIM approach works with real building elements. These elements behave according to their function in the building. For example, a wall has height, thickness, finishing and other non-geometrical properties such as cost, fire-resistance rating, compression strength, and so on, all of which are established through its parameters.

Parametric Modeling tools, also called generative design tools or semi-open platforms [14], are different from the CAD systems and BIM tools and are quite flexible computational applications. Within them, the user is able to create his own geometry with parametric rules through scripts that are visually supported. Characteristically, parametric relations among the model's elements are shown in a symbolic diagram. These tools are mostly used to work with conceptual design. During this phase, they allow for the creation of extremely complex geometry. With these tools, the user initially spends more time establishing the parametric rules in the model. However, once these rules are captured, the model's manipulation and the creation of alternative designs are easy and rapid. Parametric design relies on the ability to change an object's properties numerically without the need to redraw [15].

Other analysis tools are used in the AEC industry; among them, the structural, energy and cost analysis tools are used in MDO research. These tools are very diverse and have evolved from isolated models to analytical 3D models. Some of these tools allow the exchange of models that hail from CAD or BIM tools, which is achieved through standards such as the Industry Foundation Class (IFC), Drawing (DWG), and Green Building Extensible Markup Language (gbXML).

1.2. Successful applications of MDO in the AEC industry

A way to improve the design process is by working with computing tools that accelerate multi-disciplinary processes. In the literature, authors consider the use of standard data schemes such as IFC, gbXML or text files to transfer information from one tool to another during the design process. Examples of the use of these standards can be found in numerous texts [16–20]. While the use of standard schemes is widely used, it only solves the exploration of design alternatives with the designer intervention to prepare files and geometric models and to operate calculation models [21]. In addition, an IFC schema is not sufficient for interoperability [22] mainly because of the inconsistency among applications [21]. Moreover, the IFC standards do not support Parametric Modeling according to Steel et al. [23], who explained that supporting Parametric Modeling into the IFC language would entail significant rework of the IFC standard.

Beyond the data schemas, the MDO concept and its methodology have been used in manufacturing and aerospace industries to optimize product design [7,8]. Few examples of its application, however, have been shown in the AEC industry because MDO is basically at the research stage in this industry. One problem to implement MDO is the complexity that involves skills such as programming and an advanced use of information technologies. These skills are uncommon among architects, engineers and tool users [3]. Recent studies in this field have been carried out by other authors. For example, Marzouk and Abubakr [24] proposed a model called TCrane_Opt that could optimize the location of a group of tower cranes in construction sites, thus minimizing the total cost of the tower crane and maximizing its utilization. They also used a BIM model to extract information for the optimization and clash detection processes. Ibrahim et al. [25] developed a model of site facilities to ensure a realistic approach to construction site layout problems. They used optimization algorithms to work with static and dynamic shapes within Grasshopper® and Rhinoceros®. Faghihi et al. [26] used a BIM model as an input to find the trade-off relationship between predefined objectives, such as time cost and job-site movements of a construction project. Dino [27] presented a novel approach to the 3D space layout problem through an Evolutionary Architectural Space Layout Explorer (EASE) tool that facilitates the optimization and exploration of 3D space layouts to satisfy constraints of space overlaps and empty areas: size, geometry placement and topology relations. Chardon et al. [28] coupled a Non Sorting Genetic Algorithm (NASGA2) to an integrated design tool via data exchange. These authors implemented optimization to minimize global construction costs and energy performance for dwellings. Ferrara et al. [29] coupled a generic optimization tool (GenOpt®) to a Transient System Simulation Tool (TRNSYS®) to study how the energy system affects the technical and economic optimal design solutions of the building in two different climate conditions.

MDO has been mainly performed through multidisciplinary processes that use heterogeneous computational tools, which are coupled to each other or to optimizing platforms. These platforms generate a single process environment, in which an optimization routine simulates analyses on geometrical models. This single process environment is usually called Process Integration and Design Optimization (PIDO), which contains several optimization algorithms that allow rapidly exploring design alternatives and performing tradeoffs in order to select them. Researchers have used PIDO for MDO research. For example, Welle et al. [16] created an optimization methodology called ThermalOpt to perform MDO with BIM tools (Digital Project) focused on energy optimization. This research coupled several simulation tools in an optimization routine within PIDO to overcome the technical barriers of the applications' separated file formats. The barriers were the inconsistences and interoperability issues among the architectural, structural and energy models. The exchange of data was based on IFC files. Basbagill et al. [30] presented a case study that integrated MDO with conceptual building design to feedback designers after decisions. They decreased the

Table 1
Summary of literature review on MDO research in the AEC industry.

Reference	MDO	Tools used						Disciplines studied							Coupling method	Tool integration challenges and strategies	Detail of the coupling	MDO requirements
		BIM	PIDO	PM tools	Structural tool	Mathematical tools	Energy tool	Structural	Energy	Architecture	Sustainable Design	Cost	MEP	Urbanism				
[3]	X	X	X	X	X		X	X	X						Macro input file, batch file	Experts for wrapping	–	Barely mentioned
[31]	X		X	X			X		X				X		Plug-in to pass information	–	–	–
[32]	X		X	X			X		X						DOS® batch file	Customized codes	–	–
[16]	X	X	X	X			X		X						Plug-ins, wrappers	Long time, interoperability issues	–	–
[17]		X		X			X		X					X	Interoperability features without wrapping	–	–	–
[18]	X			X			X		X						Text file	–	–	–
[33]	X		X	X			X		X			X			–	–	–	–
[34]	X			X			X		X		X				–	–	–	–
[35]	X			X			X		X						–	–	–	–
[36]	X		X		X			X							–	–	–	–
[37]	X	X		X											–	–	–	–
[38]	X		X	X			X		X						–	–	–	–
[20]	X	X		X					X		X				Script file, batch file	–	Generically mentioned	–
[19]				X			X		X						Text file	–	–	–
[39]	X					X				X					–	–	–	–
[40]	X				X			X							–	–	–	–
[30]	X	X	X				X		X	X		X			–	–	–	–
[41]	X			X				X		X	X	X			–	–	–	–
[42]	X	X					X		X	X		X			Plug-in	Interoperability issues	–	–
[43]	X					X	X		X		X	X			–	–	–	–
[44]	X				X			X		X		X			–	–	–	–
[45]		X		X			X								–	Interoperability challenges	–	–
[24]		X				X						X			–	–	–	–
[25]	X	X		X						X					–	–	–	–
[26]	X	X						X				X			–	–	–	–
[27]	X					X				X					–	–	–	–
[28]	X	X					X				X	X			Ontology rules for databases	–	–	–
[29]	X		X				X		X			X	X		Manual editing of input text files	Issues with the existing wrapper	–	–

carbon footprint and showed that the automated sequential feedback method could accommodate multiple objectives. The optimization routine was implemented by integrating seven software components to a PIDO platform.

A summary of our literature review on recent MDO research in the AEC industry is shown in Table 1, which summarizes the tools used by the authors, the disciplines involved in the optimization study, the coupling method, the challenges and strategies for tool's integration, the detail of the coupling and MDO requirements.

1.3. Research problem and objective

Tool integration is a crucial issue in MDO. As can be observed in Table 1, it involves working with heterogeneous software that has been developed by different people at different times, using different methods, depending on the type of tool. Furthermore, these tools may have different functionality, scope and mode of interaction with the user. While these tools have been coupled in MDO research, only some authors have mentioned some of the coupling methods such as Batch Processing (BP), file scripting or the use of plug-ins to import and export information. Several authors e.g., [3,16,31,42] used plug-ins to couple the tools to a PIDO environment, while other authors e.g., [3,20,32] used BP and text files as the coupling method. Negendahl [45] mentioned three methods of coupling between tools: combined model, central model and distributed model, of which only the distributed model method refers to the integration of computational tools. Flager et al. [3] mentioned that the only requirement is that the tool runs in Batch Mode, however, this is only one method for coupling tools. The authors did not explain what happened during the coupling process in terms of the technological implications of the integration, the tool's behavior and their requisites for coupling or the best strategies and challenges experienced in integration. Although various authors e.g., [16,42,45] faced interoperability challenges when exchanging data and format issues, these challenges referred to an imprecise data conversion and the long time spent preparing models, but not to tool integration. There may be additional technical requisites for MDO, other than coupling, that have not been studied in the literature.

In fact, the basic requisites to develop MDO using PIDO platforms and the challenges that designers can face during its implementation are not clear. Finally, design optimization literature has revealed only few strategies to develop MDO, which provide little help for its use and implementation by AEC companies. Therefore, if MDO is to be implemented in the AEC industry, these challenges and strategies need to be elaborated.

In summary, while there are significant contributions to MDO research; there is only a limited description of the technical requisites to perform it through PIDO. There is no clear explanation of tool capabilities compared to these requisites, the coupling methods for these tools with PIDO, the general considerations, the challenges faced, or the strategies to follow.

The specific findings obtained by the MDO literature review were as follows:

- The basic requisites for developing MDO using PIDO are scarcely mentioned. Although both part of the technical requisites for MDO and other requisites regarding the optimization process have been mentioned, further research is necessary to understand how MDO functions in PIDO platforms
- Only some researchers have utilized optimization platforms, while other researchers integrate the design process by exchanging data through standard files such as IFC or gbXML
- The coupling strategies among ITs are not explained in most articles nor are the capabilities of the software tools
- The researchers have failed to explain the challenges faced and the capabilities that tools must have.
- In some cases, there is a lack of tool integration in the optimization processes

The current literature does not solve these issues and still leaves these questions: What are the technical requisites for basic tools to perform MDO using PIDO? Could any AEC tool work with a PIDO platform? What challenges will be faced when working with PIDO? What strategies should be used in order to work with PIDO?

The contribution of our work is to answer the above questions and test our hypothesis: that any AEC tool can work within PIDO to develop MDO. First, our findings should help design researchers to simplify the process of tool selection and second, they also will help researchers to be aware of the basic aspects to consider when working on an MDO framework.

This paper is structured as follows: first, it describes the methodology employed to determine basic tool requisites to perform MDO using PIDO, evaluate the AEC tools against the technical requisites and simulate a design process within PIDO. Later, it presents the main results divided into five sections: the tool requisites to develop MDO with PIDO, the behavior, strengths and weaknesses of the AEC tools versus the technical requisites, the challenges faced, and general successful strategies to develop work with PIDO. Finally, it offers a conclusion and recommendations for future work.

2. Research methodology

One way to develop MDO is through the use of PIDO tools linked to other conventional tools. However, in the literature, the technical tool requisites to work with PIDO are not clear. The main purpose of this research is to determine these requisites, demonstrate the tool capabilities with respect to these requisites, and demonstrate that tools work in a design simulation routine within PIDO before performing optimization. Consequently, a three-phase methodology was designed and is explained in this section (Fig. 1).

2.1. Phase one

In order to work from PIDO, it was necessary to know the capabilities of the tools. One way to meet these requirements was by directly

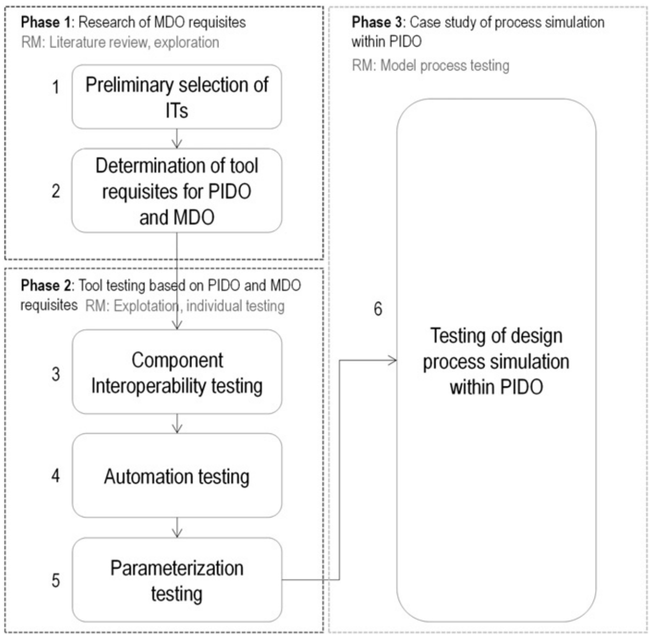

Phase 1: Research of MDO requisites
RM: Literature review, exploration
1
Preliminary selection of ITs
2
Determination of tool requisites for PIDO and MDO
Phase 2: Tool testing based on PIDO and MDO requisites RM: Explotation, individual testing
3
Component Interoperability testing
4
Automation testing
5
Parameterization testing
Phase 3: Case study of process simulation within PIDO
RM: Model process testing
6
Testing of design process simulation within PIDO

Fig. 1. Three-phase research methodology. RM: Research methods.

selecting and exploring the tools and also reviewing literature. Therefore, this phase consisted of two steps: preliminary selection of AEC tools and determination of tool requisites for PIDO and MDO. For the preliminary selection, we sought and initially selected five tools based on two criteria: initial features and academic availability. There were three types of tools: two of them belonged to BIM: Autodesk® Revit® and Bentley® AECOsim Building Designer™; two belonged to structural analysis tools: Bentley® StaadPro® and Autodesk® Robot™, and the last of them was a PM tool: Bentley® GenerativeComponents®. The PIDO platform was ModelCenter® from Phoenix Integration®. Our selection was based on the tool features that were grouped according the type of tool. These features are shown in Table 2. Fortunately, these tools were readily available them by means of academic agreements. Once the tools were selected, we could fully explore them. Additionally, the determination of tool requisites for PIDO and MDO permitted the evaluation of the pre-selected tools according to these requisites. The results of this phase are explained further in the results section.

2.2. Phase two

The second phase was not determined until the MDO requisites – component interoperability, automation of tool tasks, and model parameterization - were identified at the end of the first phase. The second phase was designed to individually test each of the five AEC tools in order to discover their capabilities based on these three requisites and to test the initial hypothesis of this research. We developed testing of component interoperability, automation and parameterization for the five preselected tools. After the testing, we finally were able to select the best tools, those that met all the requisites.

2.3. Phase three

The purpose of this phase was not to develop MDO yet, but to show that the tools finally selected worked together in a design simulation routine within PIDO. For this purpose, a process model was created within PIDO in order to observe and validate that the process started; the tools performed their corresponding analysis; data were transferred from one tool to another; tools interpreted the data transferred and used them correctly; and finally, the expected outputs were produced by the simulation. This process involved working with three tools in one environment: one design tool, one structural tool and a PIDO tool. The process was applied to a concrete frame involving architectural dimensions of the frame, structural analysis and cost disciplines. Then a change on the frame's dimensions was applied in order to observe the coordination of these disciplines within the process model.

Table 2
Features for initial selection according to tool type.

Type of tool	Features considered for initial selection
BIM tools (Revit® and AECOsim Building Designer™)	* contain architectural, structural, mechanical and electrical disciplines * can create IFC files * have APIs * have embedded analyses such as clash detection * have links with structural analysis tools
PM tool (GenerativeComponents®)	* converts own models to IFC and CAD formats * has an API * can integrate with AECOsim Building Designer ™
Structural tools (Robot®, StaadPro®)	* have direct links with their corresponding BIM tools (Revit®-Robot™; AECOsim Building Designer™-StaadPro®) * have APIs
PIDO platform (ModelCenter®)	* has a variety of optimization algorithms for MDO * is supported by a technical team * has different wrapping methods available * has an API

3. Results and discussion

MDO is important in the AEC industry. However, the lack of explicit technical requisites has made it difficult to implement through PIDO by integrating modeling and analysis tools. This challenge motivated us to complete the three-phase methodology and obtain results on MDO requisites and tool evaluation. This section will present these results in five parts. First, there is a description of the tools' required capabilities to achieve MDO: component interoperability, tool automation, and model parameterization. Second, there is a summary of the behavior, the strengths and weaknesses of AEC tools studied according to the MDO requisites in the methodology's second phase. Third, a design process model within a PIDO platform to simulate the multidisciplinary interactions among the architectural, structural and costing disciplines is shown. Fourth, the challenges to meet the requisites of MDO are clarified. And finally, we propose some strategies to successfully prepare a workable environment for MDO.

3.1. Results of phase one: description of the tools required capabilities for MDO

Working with PIDO requires its integration with different tools through a means of communication that is not clear in the MDO literature. In fact, tool exploration was often necessary to discover the other requisites to perform MDO; and we also needed to examine more literature from computing science. For the literature review, we studied articles on interoperability; we also searched for tool integration and modes of interaction with the tools in their official forums, and searched for wrapping methods in the documentation of PIDO. From these searches, we could finally summarize fragmented information regarding interoperability and tool integration. In addition, in order to explore the AEC tools, we searched for manipulation methods of their elements in 3D models and learned how to perform specific analyses, how to work with associative relationships among elements of a model, and finally, how to control the tools without human intervention. As a result of this step, we defined three technical requisites for the tools: component interoperability, tasks and analyses automation, and model parameterization: capabilities that the tools must meet. This section discusses these capabilities in detail.

3.1.1. Component interoperability

After reviewing the interoperability literature, as the first result, we found that the term "integration" has a technical synonym: "component interoperability". This concept is one of the key elements for MDO and is essential for integrating software components with PIDO platforms. It could be understood as the ability of two or more entities to communicate and cooperate with each other without considering the implemented language, execution environment, and model abstraction differences [46]. For us in the AEC context, it is the integration of separately developed software components so that they can interact. PIDO platforms work within a component interoperability framework to integrate computational tools such as modeling and analysis tools. Component interoperability is developed through the applications' interfaces: Application Programming Interface (API) and/or BP but they must be controlled externally: from within third-party applications and/or Windows® Command Line Interface (CLI), respectively. These two types of interfaces are discussed in this section.

3.1.1.1. Component Interoperability through API. The tools are developed by different people and have different API types. Each API type depends on the middleware technology used. Middleware is based on various standards that allow component interoperability. These standards have given rise to open APIs. This subsection is divided into five paragraphs that explain the concept of middleware, its purpose and elements, types of middleware and architecture technologies, and its interfaces.

Middleware is a key element that makes component interoperability possible through an API [10,45]. Middleware has been defined in at least three ways. According to Madiajagan and Vijayakumar [46], it is a universal communication bus and the "glue" of any information system. It also refers to a software set that allows communication and information exchange between client and server components, as well as organizing such exchange [46]. Additionally, it is a software layer that is located between the operating system and the application. It provides reusable well known solutions to frequently occurring problems such as heterogeneity, interoperability, security, and dependence.

The main purpose of middleware is to overcome the heterogeneity of distributed infrastructure by a well-defined and structured programming model [45,47] (Fig. 2). This structured programming model defines five elements [46,47]:

- An Interface Description Language (IDL) that is used for specifying data types and interfaces of networked software resources
- A high-level addressing scheme based on the underlying network addressing scheme for locating resources.
- An interaction paradigm and semantics for achieving coordination
- A transport/session protocol for achieving communication among components
- A naming/discovery protocol, naming/description convention, registry structure, and matching relation for publishing and discovering the resources available in the given network.

Issarny et al. [47] mentioned five types of middleware: transactional, tuplespace-based, remote procedure calls, object-oriented and service-oriented. According to these authors, object-oriented and service-oriented middleware apply to component interoperability. These types of middleware have evolved from remote procedure calls (RPC) to the Object-Oriented Programming (OOP). They permit the user to reference remote objects and call operations in them [48]. We focused on these two types of middleware. They rely on standards that are the foundation of interoperability. According to these standards, there are several types of architecture technologies that define Object-Oriented Middleware [49]:

- COM™, DCOM™, COM™+: The Component Object Model (COM™) [50] from Microsoft® is strongly relied on the Microsoft®'s operating systems. COM™ components are declared using a Microsoft® IDL (MIDL®) that supports the description of COM™ components classes and interfaces. Unlike CORBA®, the interfaces only define methods. Properties are declared using get and set methods. These methods define special attached attributes. MIDL utilizes its own type of system based on C system. Components are usually implemented by C++ classes or by another language that supports COM™.
- CORBA®: Common Object Request Broker Architecture (CORBA®) [51] provides objects' infrastructure that can interoperate among different software and hardware products. A CORBA® object is declared by writing an IDL file. This file contains the object's interface definition. This interface takes the definitions of an object's operations and attributes. The IDL file is compiled by an IDL compiler that generates client stubs and server skeletons for a given language. IDL has its own system which is broadly based on C++. The model communication of CORBA® is based on object invocation. These objects can be local or remote.
- EJB™: The Enterprise JavaBeans™ (EJB™) [52] is a server-side component architecture. The purpose of EJB™ is to simplify the creation of distributed component applications in Java. These applications can be coded without writing a complex distributed component framework. EJB™ supports portability and reusability of applications across any middleware services. The EJB™ implementation consists of Java classes that are deployed in a container on an application server. Clients use an enterprise bean's home and remote interface to invoke its methods. The home interface defines methods to create or to look up component instances. The remote interface provides access to a given instance.

An interface is a collection of possible functions to specify a software unit's services through its operations [46] and is a key element for middleware technology. Depending on this technology, the interfaces are developed with a specific IDL such as OMG® IDL for CORBA®, Microsoft® IDL for COM™, Java™ interface for Remote Method Invocation (RMI), and Web Services Description Language (WSDL) for web services, and all interfaces have distinct degrees of openness: closed and opened. Closed APIs are limited only to certain functionality. This functionality depends upon the developer. In some cases, they do not allow external control of their application. In contrast, the development of open APIs is based on open standards that permit access to greater application functionality which can be externally controlled. Computer-aided process engineering (CAPE)-OPEN is a key example of an open interface and standard technology for interoperability and engineering process integration with software components. CAPE is an abstract specification that can be subsequently implemented in COM™, CORBA®, and NET™ to bridge tools. It allows one to achieve interoperability and to wrap tools with third party software. This is a standardization effort to achieve actual connections (plug & play) among software components of processes simulation and environments [53].

3.1.1.2. Component Interoperability through Batch Processing. Batch Processing (BP) is another type of interface to achieve component interoperability. It is still active in many applications and is the execution of a tool without human intervention for repetitive and large amounts of information. BP is developed through scripts. The next two paragraphs explain the BP concept and its functionality.

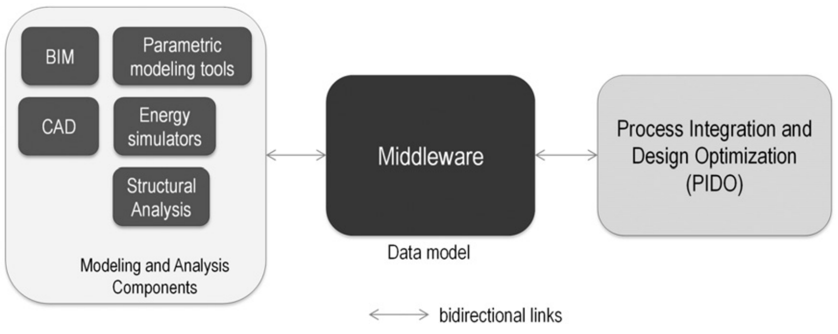

BIM
Parametric modeling tools
CAD
Energy simulators
Structural Analysis
Modeling and Analysis Components
Middleware
Data model
Process Integration and Design Optimization (PIDO)
bidirectional links

Fig. 2. Middleware overcomes heterogeneity among tools (Adapted from [45,47]).

BP or Batch Mode (BM) means executing an application with minimal or no Graphic User Interface (GUI). Hence, when running, it does not request data entry or directions from the user. BP is used to automatically develop repetitive tasks and to process large amounts of information in order to avoid human errors and save time. Repetitive tasks are usually tedious, time consuming, and performed on several steps. Conversely, large amounts of information become unmanageable if performed manually.

The functionality of BP depends on the tool. In some tools, it is used to print and update file versions; but in others, it executes scripts of commands and macros. Scripts are generally text files that contain instructions and can be run from within other applications. These instructions can be predefined lists of commands or complex macros. Sometimes, scripts are text files that contain recorded user actions that create the geometric model and can be modified and then run via BP to perform an analysis or implement a change in a model. They also can be run from within other applications to perform component interoperability.

3.1.2. Tool Automation

In many computing tools, the user needs to click icons and enter data manually to perform tasks using the GUI. In an MDO context, in contrast, there is no human intervention when developing a repetitive workflow to create and analyze design scenarios. Hence, we found that tool automation was another requisite for working in a MDO framework. For some authors, component interoperability and automation were part of the concept “integration”. This research, however, considered them as independent concepts because a tool can be automated but, at times, this automation cannot be controlled externally. Automation can be achieved using a workflow executed automatically; in which the tools must be automated separately by distinct methods: a command line, text files and/or macros. These methods are tightly related to the tools' interfaces and this section explains them in the following three paragraphs.

Tools that have a command line interface at times can execute chains of commands simultaneously. It permits the performance of specific actions on certain components in a model or triggers a general analysis such as: 4D, clash detection, structural and so on by pressing only one key. It avoids use of the GUI and allows commands to be manipulated using BP or an API.

Some tools do not have a command line interface but they possess text files that record all or most of the user actions. These files can be modified in a text editor and then run via BP as arguments after the execution path of the application. This method permits making modifications on a 3D model and performing analyses.

Additionally, there are tools with a macro editor. The macro editor permits the application to perform analysis and execute repetitive actions automatically. The macro editor allows writing and/or recording a sequence of actions performed by the user with the GUI. This sequence can be adapted programmatically to meet specific functions. At times a macro can run, and arguments can be passed to the macro through the command line interface.

3.1.3. Model Parameterization

After experimenting with tools and reviewing the literature, we found that parameterization is another key requisite to develop MDO, and it can be achieved through Parametric Modeling. Parametric Modeling implies change [15]. It enables the designers to create complex forms and designs with precise control by establishing hierarchical associations among discrete parts [15] and to create and explore design options from a “parametric model”. In MDO, the design options must be made coherent by means of the relationships and connections among all of their parts because no human intervention is present to adjust the relationships of a model when the optimization routine is executed. For coherence in design options, tools must have four Parametric Modeling capabilities: parameters' creation and modification, inclusion of complex logic and rules in parameters, creation of associations among components by their parameters, and creation of custom parametric components. These capabilities are explained in this section.

Tools should allow the creation of both general and specific parameters that represent a model's information. General parameters belong to the whole model and can represent a general behavior. In contrast, specific parameters belong to a particular model's components and can affect both the overall model as well as other component parameters. These two types of parameters can store geometrical or non-geometrical information. Geometrical information refers to dimensions of components and their parts. Non-geometrical information refers to characteristics such as strength, cost, specifications, and insulation rate.

Parameters should permit the user to store not only a single value but also complex logic to determine a series of values depending upon input values of other parameters. This logic is referred to programmatic decision rules. These rules help determine a parameter's value and automatically coordinate model changes.

The creation of parametric associations among elements is crucial to maintain model coherence. Coherence refers to the logical, well-organized and harmonic associations among a model's components. It is necessary when an optimization algorithm modifies the value of some parameters that directly affect the geometry or other properties of particular components and when they are associated with others. For example, if a wall is connected to a slab, when the wall's length changes, the slab's span must change as well.

Finally, the creation of custom parametric content is required because there can be many components and building parts not included in the parametric families of the tools. Moreover, the existing components might not contain all of the parameters and behavior required by the designers.

3.2. Results of phase two: behavior of the five AEC tools in achieving MDO

Upon concluding phase two, we discovered that the five tools tested behaved differently and had strengths and weaknesses in terms of the three requisites established for MDO: component interoperability, tool automation, and parameterization. Some AEC tools did not possess full capabilities to work within PIDO. This section explains the behavior, strengths and weaknesses of these tools.

3.2.1. Behavior of the AEC tools tested against MDO requisites using PIDO

Of the five tools studied – AECOsim Building Designer™, Revit®, GenerativeComponents®, StaadPro®, and Robot™ – we selected only two tools for working with ModelCenter®: GenerativeComponents® and Robot™. This selection was based on the different behavior and distinct strengths (Section 3.2.2) of these tools in terms of component interoperability, tool automation and parameterization. Some had a broader scope than others because they had more capabilities in each of the three categories. Others had limited or no capabilities in some of the categories such as tool automation and parameterization, respectively. These differences are explained in each following testing.

3.2.1.1. Component Interoperability testing. There were four tests applied to the tools in order to know whether they could be integrated into the MC. These tests included searching for the type of non-graphical interface and programming language, verifying the external control of the tool, evaluation of the degree of openness of the tool interfaces and seeking the most efficient wrapping method for each tool. Table 3 states the purpose of these tests.

The tools had different interfaces and programming languages, degrees of interface openness and integration methods with PIDO. Each tool had an API, however, these APIs were based on different programming languages: Visual Basic for Applications® (VBA®), Visual Basic® (VB®) and C#®. Only AECOsim Building Designer™ and GenerativeComponents® consisted of a supported BP interface. Revit® included a Journal mode similar to BP but it was not

Table 3
Purpose of the type of tests for Component Interoperability.

Type of testing for Component Interoperability	Purpose of the testing
Search and exploration of non-graphical interfaces	To explore the non-graphical interfaces, their documentation
Interface openness degree	To know the scope of the tool interfaces
Control of the tools from outside	To control tools from within third-party applications by means of the use of scripts, macros or Batch Processing
Wrapping method for PIDO	To seek the best method for coupling each tool to PIDO

supported by Autodesk®. These interfaces differed in their degree of openness, meaning that the greater the openness, the greater the external control of the tool. For example, it was possible to control AECOsim Building Designer™, StaadPro® and Robot™ from within Microsoft® Excel ® (Excel®) because they had open APIs that enabled communication with third party applications. However, it was not possible to fully control externally either Revit® or AECOsim Building Designer™ through BP because these interfaces were almost closed and did not allow access to much of their functionality. Finally, the tools enabled wrapping to PIDO by different methods. AECOsim Building Designer™, StaadPro® and Robot™ were wrapped by a script written in Excel® or in the ModelCenter®'s component script. However, GenerativeComponents® was wrapped by BP because its API was closed (Table 6).

3.2.1.2. Automation testing. To automate actions and commands, six automation tests were identified by exploring the tools or by studying tools' documentation. The tools were evaluated with respect to these tests: the existence of command line interface, the possibility of running silently, the execution of tasks with only one action, the existence of text scripts, the existence of macro editors and the possibility of passing arguments from the command line to a macro. The purpose of these tests is shown in Table 4.

The tools studied were automated by different methods, some of which were more complex than others, and were built by assorted capabilities such as command line interface (CLI), chains of commands, script text files, macro editors and the quality of passing arguments via CLI to macros. Only AECOsim Building Designer™ and StaadPro® had a CLI. AECOsim Building Designer™'s CLI was internal and explicit while StaadPro®'s CLI was externally used through Windows® CLI. From the five tools, AECOsim Building Designer™ allowed writing chains of commands, which aided in the concatenation of commands to automatically perform several actions on components simultaneously. For example, moving a component and changing its properties simultaneously could be performed by joining commands. Another capability was the use of the script text files, which stored a series of commands and actions and were run via BP. They appeared in different forms according to the tool. For instance, in AECOsim Building Designer™, they were batch files; in Revit®, they were journal files; and in GenerativeComponents®, they were transaction scripts. Sometimes, they stored and translated a complete user session via the GUI of a tool into written actions that recreated the whole session. Most of tested tools had a macro editor, which was the most standardized method to automate a tool. Macros were used to design and execute custom actions that did not exist in a tool via preset commands. These macros were written in VBA® or C#® programming languages. Finally, values were able to be passed to macros as arguments when macros ran from CLI. These values served as input parameters, conditioned the outputs of a macro's process, and were only present in AECOsim Building Designer™ (Table 6).

3.2.1.3. Parameterization testing. Parameterization is essential so that both PIDO accesses to a model's parameters to manipulate it and design options are coherent when generated automatically. These parameters can be input or output variables that show a result. Parameterization consisted of five tests that were mainly applied to BIM and PM tools. These tests were the creation of general parameters, the creation of specific parameters, the possibility of embedding complex rules within parameters, the possibility of parametrically associating components and the creation of customized parametric components. The purpose of these tests is shown in Table 5.

The five tools had distinct capabilities of parameterization. As for the creation of general parameters, GenerativeComponents® was the most powerful tool for creating them because it enabled the creation of global variables. Structural analysis tools allowed the creation of these parameters programmatically. BIM tools did not work with general parameters. Generally, all of the tools made possible the creation of component-specific parameters. These parameters determined the properties and behavior of specific project's components. In addition, GenerativeComponents® supported embedding a complex logic in parameters to determine their value. These parameters were calculated according to their rules and made a dynamic model for changes. Structural analysis tools allowed embedding complex rules and logic through programming as well. In BIM tools, embedded rules were possible in the creation of component families. Parametric associations among components were enabled in GenerativeComponents®, StaadPro® and Robot ™. These associations helped to maintain coherence when changes were applied to models. Finally, in GenerativeComponents®, it was possible to create bespoke parametric content either graphically or programmatically. BIM tools permitted the creation of certain types of families of accessories but not system families such as walls, beams,

Table 4
Purpose of the type of tests for Automation.

Type of testing for Automation	Purpose of the testing
Existence of Command Line Interface	To run preset commands without using icons
Possibility of running silently	To run a tool with its GUI closed and avoid pop-up windows which ask for user data input
Execution of tasks with only one action	To perform actions using chains of commands whose inputs are given simultaneously and avoid pop-up windows
Existence of text scripts	To automatically run script text files which can contain a series of commands
Existence of macro editors	To automate commands and actions by means of writing or recording a macro that executes them
Possibility of passing arguments from the command line to macros	To pass input data as arguments to macros and use them within the macros' code

Table 5
Purpose of the type of tests for Parameterization.

Type of testing for parametrization	Purpose of the testing
Creation of general parameters	To create general parameters that belong to the whole model
Creation of specific parameters	To add new parameters to components in order to manipulate them in a 3D model
Possibility of embedding complex rules within parameters	To incorporate complex mathematical expressions into the parameters of components in a 3D model
Possibility of parametrically associating components	To parametrically link components to maintain coherence in a 3D model when implementing changes to it
Creation of customized parametric components	To create new customized parametric components that are not included in commercial tools

Table 6
Behavior of AEC tools tested against the technical requisites of MDO through PIDO * Through Excel® and VBA® programming.

Tool	Component Interoperability			Tool Automation						Parameterization				
	Type of interfaces and programming language	Interface openness	Wrapping method for PIDO	Command line interface (CLI)	Silent mode	Chains of commands	Script text files	Macro editor	Arguments passed by CLI to Macros	Model general parameters	Specific component parameters	Complex rules embedded in parameters	Parametric associations among components	Custom parametric families
AECOsim Building Designer™	API (VBA®) Batch Processing	Open Closed	Component Script, Excel plug-in	Yes	No	Yes	Yes	Yes	Yes	No	Partial	Partial	No	Partial
Revit®	API (VB®, C#) Journal mode	Semi-open Semi-open	Plug-in (tested by other authors)	No	No	No	Yes	Yes	No	Partial	Partial	Partial	No	Partial
GenerativeComponents®	API Batch Processing	Closed Open	QuickWrap™	No	No	No	Yes	No	No	Yes	Yes	Yes	Yes	Yes
StaadPro®	API Batch Processing	Open Open	Component Script, Excel plug-in	Yes	No	No	No	Yes	No	*Yes	*Yes	*Yes	*Yes	Not applicable
Robot™	API	Open	Component Script, Excel plug-in	No	Yes	No	No	Yes	No	*Yes	*Yes	*Yes	*Yes	Not applicable

columns, slabs, doors, windows, Mechanical-Electrical-Plumbing components and others (Table 6).

3.2.2. Strengths and weaknesses of the AEC tools tested according to MDO requisites

Since AEC tools tested did not fully meet the requisites to work within a PIDO platform and therefore to develop MDO, GenerativeComponents® and Robot™ were selected because they met most of these requisites. This research identified some strengths and weaknesses concerning the five tools studied so that future researchers may consider them. This subsection discusses these advantages and disadvantages divided into Component Interoperability, tool Automation and Parameterization categories.

Most of the tools worked well when coupled to ModelCenter® and when their models' variables were accessed from within the ModelCenter®'s GUI through either API or BP. Those tools that had an open interface, allowed full control from within PIDO. Nevertheless, a common weakness was the lack of official documentation about driving the tools from outside. Although BIM tools had two interfaces, only their APIs worked appropriately because BP was deficient in one of the tools and unsupported in the other tool (Table 7).

Automation was performed in different ways depending on the tools' characteristics. AECOsim Building Designer™, for example, had many resources for automation: CLI, macros, arguments passed to macros via CLI and complex sequences of commands. GenerativeComponents® included a transaction text file that contained all of the actions to recreate the geometry and the model's parameters. However, for Revit®, StaadPro® and AECOsim Building Designer™, automation was not possible because both tools entered their interactive mode and requested user entries through pop-up windows. This impeded a continuous workflow and demanded user interaction with the process. Overall, this failure was because these tools were not able to run silently. In other cases, it was not possible to automate all of the commands necessary to perform tasks and analyses. (Table 8).

With regards to parameterization, the tools had very different capabilities. PM and structural tools were able to fully parameterize models by creating general and specific parameters, embedding complex rules in them and associating the parameters' values. However, BIM tools presented some disadvantages because their parametric functionality varied from one tool to another. For example, some components in certain BIM tools behaved differently than those similar components in another BIM tool because they had dissimilar embedded parametric rules. Additionally, some embedded preset rules made the relations among some components incompatible. In AECOsim Building Designer ™, there were no associative rules among certain components and therefore they remained unlinked. In these components, it was not possible to create general project parameters such as general dimensions, total cost, and it only was possible to obtain 2D parametric drawings. Finally, Revit® and AECOsim Building Designer™ did not enable the creation of system families of components (Table 9).

3.3. Results of phase three: the case study of tool integration and the simulation process model within PIDO

The simulation of the process model within PIDO with tools that met the requirements in the case study gave significant results. This section presents the criteria for the final selection of tested tools. Later, the integration of the two selected tools with PIDO and a diagram of the design process simulation are presented. Finally, the results of the execution of this simulation process are shown.

Only four of the five tools studied could be integrated to ModelCenter® through their interfaces. AECOsim Building Designer™, StaadPro® and Robot™ were integrated via their API. GenerativeComponents® was integrated through BP. Revit® could not be integrated to ModelCenter® because it was not possible to externally control the tool via any interface. However, although

Table 7
Tool capabilities in Component Interoperability.

Component Interoperability capabilities			
Tested tools		Strengths	Weaknesses
BIM tools	AECOsim Building Designer™	*Open API facilitated external control *Batch Processing ran each command internally *Variables or parameters were accessed from the API	*There was little documentation on external control *Batch Processing only worked internally but not from Windows CLI *GUI had to be open all the time
	Revit®	*Text script files (Journal files) ran externally from Windows CLI *Semi-open API allowed controlling certain Revit's actions externally *Allowed accessing the DataBase from API	*Journal files did not enable running all of the functions or provide access to the parameters *Batch Processing was officially unsupported *API did not allow running all of the functions and actions *There was little official documentation on external control *Access to the database was needed to manipulate a model
PM Tools	GenerativeComponents®	*Batch Processing was implemented externally *Text files allowed accessing to any parameter *Windows Communication Foundation® (WCF®) interface allowed external control	*There was very little and poor documentation on its API * There was no official documentation on external control of interfaces *Batch Processing was time-consuming when opening and closing GUI *If controlled with WCF®, GUI had be open all the time
Structural Tools	StaadPro®	*Open API facilitated external control *Command line interface controlled the tool externally	*The GUI had to be open while the tool was controlled from outside *There was an API documentation but it was incomplete
	Robot™	*Open API facilitated external control *GUI was closed when Robot was controlled from outside *Sufficient documentation on its API was available	Great complexity of the classes and the excessive amount of code

four tools were successfully integrated with ModelCenter®, other problems related to automation and parameterization arose. These problems impeded the use of StaadPro® and AECOsim Building Designer™ with ModelCenter®. In StaadPro®, for example, an inevitable pop-up window appeared. This window asked for user data input to obtain the structural analysis results and interrupted the process automation. AECOsim Building Designer™ was not able to generate parametric association rules for model components; so when making a change to the model, the components disassociated each other (i.e. a slab became disassociated from its supporting walls) and the model lost coherence (Table 10).

Finally, GenerativeComponents® and Robot™ were the two tools selected to create a process model in ModelCenter®, which illustrated that the tools can be integrated to work together within PIDO (Fig. 3).

To work with this diagram, individual work with tools was developed according to their type and a process model was created within ModelCenter®, which included the two tools.

The following tasks were performed with GenerativeComponents® (PM tool):

- Parametric components were created
- Cost design and structural analysis intelligence were embedded within components, respectively
- A model of a structural frame consisting of a beam and two columns was created
- Parametric associations for the beam and the columns were established
- A general parameter called "Total Direct Cost" for the model was created

Table 8
Tool capabilities in Automation.

Automation capabilities			
Tested tools		Strengths	Weaknesses
BIM tools	AECOsim Building Designer™	*Enabled to automate many commands and macros via CLI *Command sequences could be generated to manage compound commands *Macros could be created to automate tasks for previously inexistent commands *Arguments could be passed from a command line to a macro	*Limited CLI only activated the command but did not execute the subsequent actions and requested user interaction *Pop-up menus impeded the automation of some important actions such as: "save as" and "export" *The tool was not able to run silently
	Revit®	*Manipulation of elements was performed through a database *Some actions could be performed by running journal files	*Journal files did not automate all of the actions performed or manipulate the parameters' value *The tool did not have a CLI *The tool was not able to run silently
PM tools	GenerativeComponents®	*Used a text file that contained transactions manipulated for automation *Only certain tasks could be automated by managing compound commands and macros	*The tool did not contain a CLI *Text files did not automate all of the actions
Structural tools	StaadPro®	*SP possessed its own commands and functions to automate actions *Commands ran from Windows® CLI	*When running automatically, pop-up windows appeared, stopped the automatic process and the application returned to the interactive mode *The tool was not able to run silently
	Robot™	*It contained commands and functions to automate actions *Commands ran from Windows® CLI *It ran silently and did not request the user interface	

Table 9
Tool capabilities in Parameterization.

Parameterization capabilities			
Tested tools		Strengths	Weaknesses
BIM Tools	AECOsim Building Designer™	*Coordinates model changed and updated views without redrawing *Allowed adding parameters to elements in a model *Contained parametric embedded rules in elements *Some parameters were able to be created programmatically and passed as arguments through the CLI	*Lacked flexibility to create general project parameters *Lacked capabilities to create system families *Allowed only the creation of parameters in 2D *Contained few embedded parametric rules in its components
	Revit®	*Coordinated model changes and updated views without redrawing *Allowed adding parameters to elements in a model *Contained a lot of parametrically embedded rules in elements	*Some parametric rules impeded the combination of certain elements *Lacked capabilities to create system families *Lacked flexibility to create general project parameters
PM Tools	GenerativeComponents®	*Had great flexibility for Parametric Modeling *Had a symbolic diagram that showed parametric relations among components *Enabled complex intelligence rules *Allowed geometric and non-geometric parameters *Allowed enumerations as parameters' values *Allowed highly complex parametric geometry *Enabled the creation of custom BIM elements both graphically and programmatically	*Did not contain preset building elements *The symbolic diagram became extremely complex sometimes *There was little official documentation to program the creation of components *Programming added visual representation to elements was complex *Solving elements unions and intersections with customized BIM elements were complex *Multidisciplinary analyses of models were complex to program
Structural Tools	StaadPro®	*It was possible to programmatically create parameters *It was possible to access the parameters via its API	
	Robot™	*It was possible to programmatically create parameters *It was possible to access the parameters via its API	

■ A script text file that contained all the commands and model information was created

The following tasks were performed with Robot™ (structural analysis tool)

■ A VBA® macro was created. This macro contained all the frame model information and the commands for the structural analysis execution
■ Input and output data were linked to the macro through its corresponding spreadsheet

The following tasks were performed within ModelCenter® (PIDO platform):

■ A wrapper to integrate GenerativeComponents® with ModelCenter® was created
■ A wrapper to integrate Robot™ with ModelCenter® was performed

The wrapper of GenerativeComponents® made it possible to write or record a script text file which contained all the transactions to create and modify the components of a 3D model. This file was externally run from DOS® command line interface as an argument after the application execution. When the file was executed, the application's GUI was opened, all the actions and the 3D geometry were recreated, and the application was closed. This BP could be integrated to ModelCenter® through an instance of the QuickWrap™ component. Within QuickWrap™, input and output files were generated, which included input and output variables. These variables were manipulated from within ModelCenter® (Fig. 4).

Robot™ allows integration by an open API which enables access to many actions and functionality. One important advantage of the Robot ™'s API is that the application can run silently without keeping its GUI open. Therefore, a VBA® macro was written within Excel® to control the tool externally. This macro contained all the commands to create the frame's structural model that included the geometry, loads and supports, needed to develop the structural analysis and to obtain results (bending moments, shear force, and reactions) as outputs. Later, this macro was integrated to ModelCenter® through an instance of the existing Excel® plug-in, in which the macro code was integrated and finally input variables and output variables were created and related to the spreadsheet in the same file (Fig. 5).

After tool integration was performed within ModelCenter®, an automatic process model for design simulation was created (Figs. 6 and 7). This process simulated the multidisciplinary interactions among the architectural, structural and costing disciplines. Within ModelCenter®, choosing the model process allowed tools to be executed sequentially, linking them to each other via their input and output variables. In Fig. 7, the process begins with an architectural model consisting of a frame, the direct cost and the structural design rules embedded in the frame's components within GenerativeComponents®. This initial model feeds both the structural model within Robot™ and a final architectural model within GenerativeComponents® with geometric and structural data (dimensions and structural load). Finally, the structural

Table 10
Main reasons for tool rejection after phase two in the research methodology.

Tool studied	Wrapped to ModelCenter?	Reason for rejection
Revit®	No	Impossibility of creating system components Impossibility of controlling the tool externally Impossibility of parameterizing 3D models
AECOsim Building Designer™	Yes	Impossibility of parameterizing 3D models
GenerativeComponents®	Yes	None
StaadPro®	Yes	Automation problem prevented results of structural analysis
Robot™	Yes	None

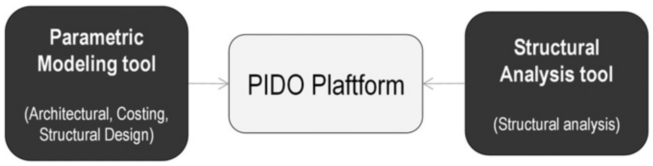
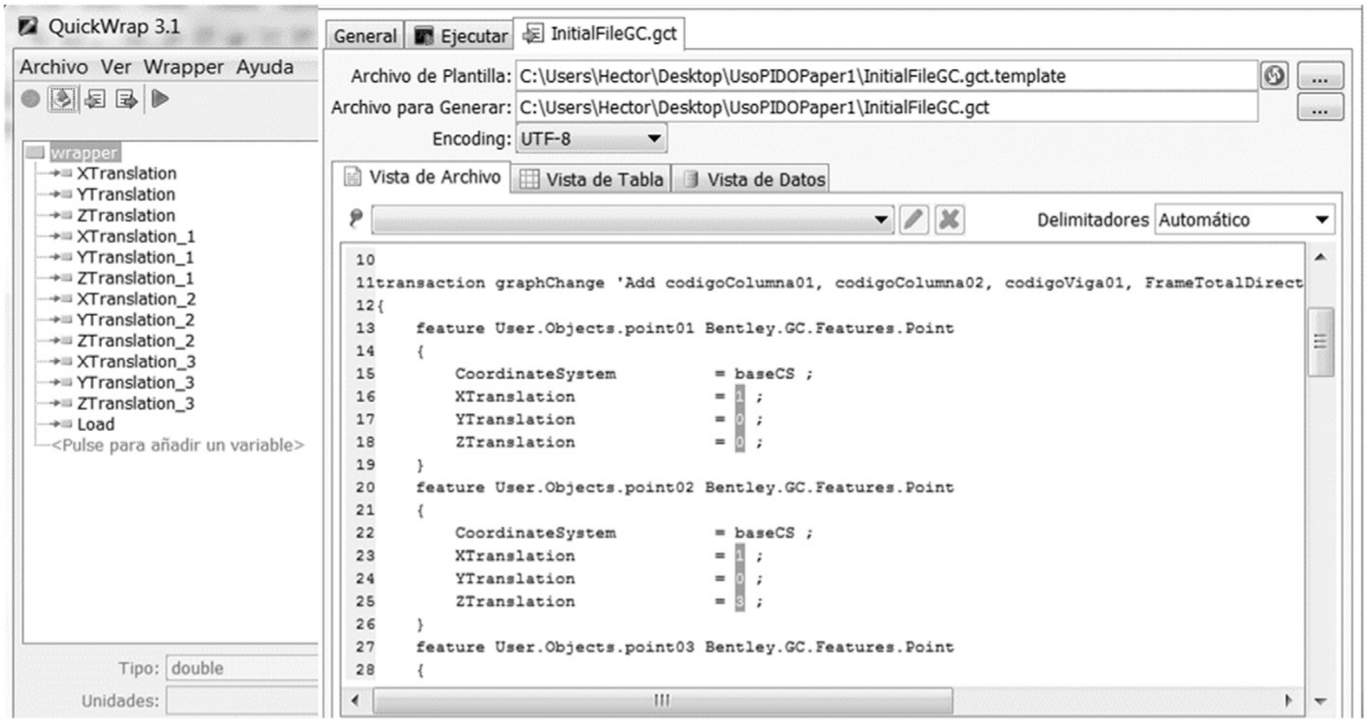


Fig. 3. Tool integration in a PIDO platform to simulate a design process.



Archivo de Plantilla: C:\Users\Hector\Desktop\UsoPIDOPaper1\InitialFileGC.gct.template
Archivo para Generar: C:\Users\Hector\Desktop\UsoPIDOPaper1\InitialFileGC.gct
Encoding: UTF-8
Delimitadores: Automático

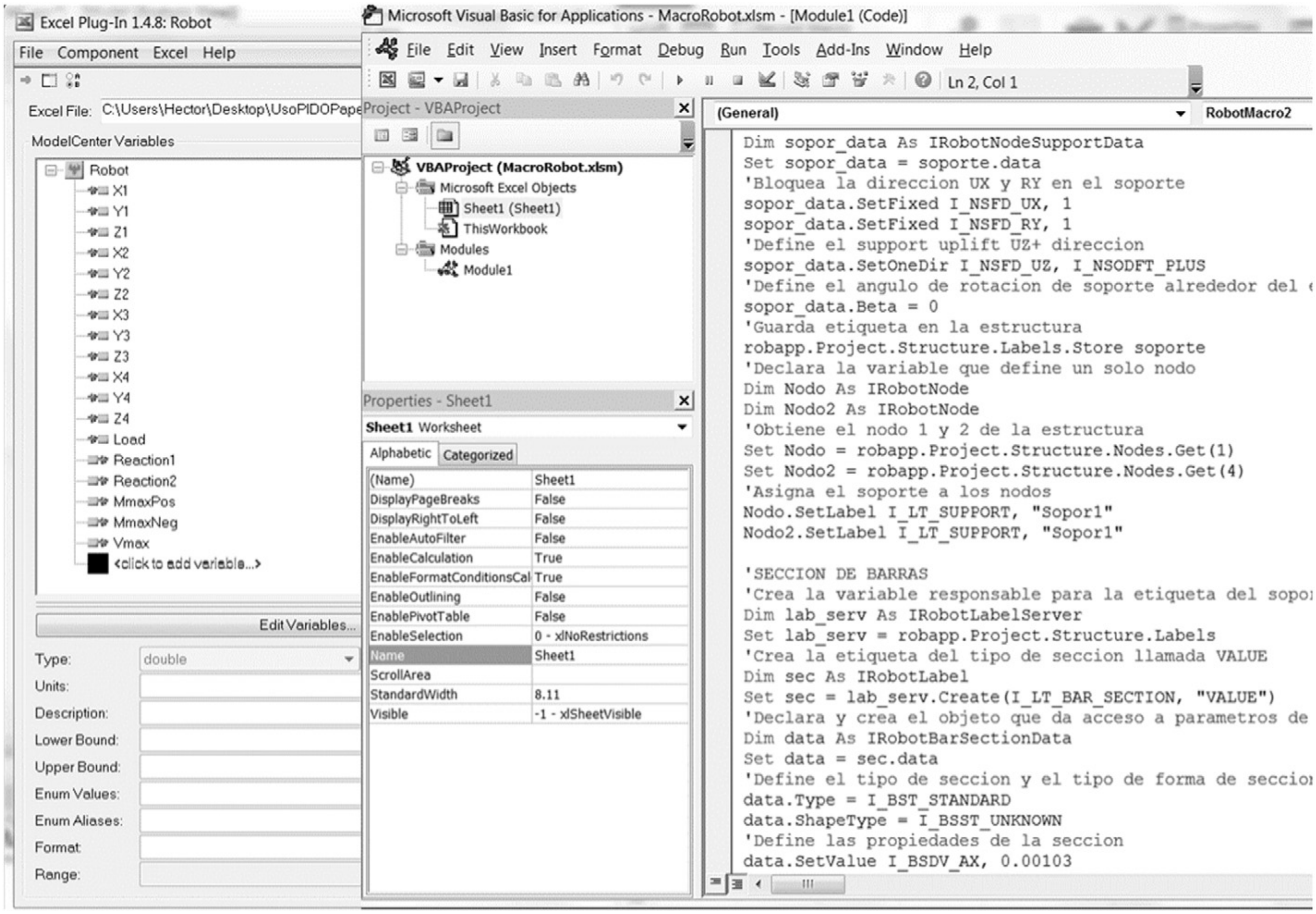
```
wrapper
  XTranslation
  YTranslation
  ZTranslation
  XTranslation_1
  YTranslation_1
  ZTranslation_1
  XTranslation_2
  YTranslation_2
  ZTranslation_2
  XTranslation_3
  YTranslation_3
  ZTranslation_3
  Load
  <Pulse para añadir un variable>
```

Tipo: double
Unidades:

```

transaction graphChange 'Add codigoColumna01, codigoColumna02, codigoViga01, FrameTotalDirect
{
    feature User.Objects.point01 Bentley.GC.Features.Point
    {
        CoordinateSystem          = baseCS ;
        XTranslation              = 1 ;
        YTranslation              = 0 ;
        ZTranslation              = 0 ;
    }
    feature User.Objects.point02 Bentley.GC.Features.Point
    {
        CoordinateSystem          = baseCS ;
        XTranslation              = 1 ;
        YTranslation              = 0 ;
        ZTranslation              = 3 ;
    }
    feature User.Objects.point03 Bentley.GC.Features.Point
    {
```

Fig. 4. Wrapper for GenerativeComponents® via Batch Processing.


Excel Plug-In 1.4.8: Robot
File Component Excel Help
Microsoft Visual Basic for Applications - MacroRobot.xlsm - [Module1 (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Ln 2, Col 1
Project - VBAProject
Properties - Sheet1
Alphabetic Categorized
Edit Variables...
(General)
RobotMacro2

Excel File: C:\Users\Hector\Desktop\UsoPIDOPape

ModelCenter Variables

```
Robot
  X1
  Y1
  Z1
  X2
  Y2
  Z2
  X3
  Y3
  Z3
  X4
  Y4
  Z4
  Load
  Reaction1
  Reaction2
  MmaxPos
  MmaxNeg
  Vmax
  <click to add variable...>
```

Type: double
Units:
Description:
Lower Bound:
Upper Bound:
Enum Values:
Enum Aliases:
Format:
Range:

```
VBAProject (MacroRobot.xlsm)
  Microsoft Excel Objects
    Sheet1 (Sheet1)
    ThisWorkbook
  Modules
    Module1
```

Sheet1 Worksheet

(Name)	Sheet1
DisplayPageBreaks	False
DisplayRightToLeft	False
EnableAutoFilter	False
EnableCalculation	True
EnableFormatConditionsCal	True
EnableOutlining	False
EnablePivotTable	False
EnableSelection	0 - xlNoRestrictions
Name	Sheet1
ScrollArea	
StandardWidth	8.11
Visible	-1 - xlSheetVisible

```
Dim sopor_data As IRobotNodeSupportData
Set sopor_data = soporte.data
'Bloquea la direccion UX y RY en el soporte
sopor_data.SetFixed I_NSFD_UX, 1
sopor_data.SetFixed I_NSFD_RY, 1
'Define el support uplift UZ+ direccion
sopor_data.SetOneDir I_NSFD_UZ, I_NSODFT_PLUS
'Define el angulo de rotacion de soporte alrededor del
sopor_data.Beta = 0
'Guarda etiqueta en la estructura
robapp.Project.Structure.Labels.Store soporte
'Declara la variable que define un solo nodo
Dim Nodo As IRobotNode
Dim Nodo2 As IRobotNode
'Obtiene el nodo 1 y 2 de la estructura
Set Nodo = robapp.Project.Structure.Nodes.Get(1)
Set Nodo2 = robapp.Project.Structure.Nodes.Get(4)
'Asigna el soporte a los nodos
Nodo.SetLabel I_LT_SUPPORT, "Sopor1"
Nodo2.SetLabel I_LT_SUPPORT, "Sopor1"

'SECCION DE BARRAS
'Crea la variable responsable para la etiqueta del sopo
Dim lab_serv As IRobotLabelServer
Set lab_serv = robapp.Project.Structure.Labels
'Crea la etiqueta del tipo de seccion llamada VALUE
Dim sec As IRobotLabel
Set sec = lab_serv.Create(I_LT_BAR_SECTION, "VALUE")
'Declara y crea el objeto que da acceso a parametros de
Dim data As IRobotBarSectionData
Set data = sec.data
'Define el tipo de seccion y el tipo de forma de seccio
data.Type = I_BST_STANDARD
data.ShapeType = I_BSST_UNKNOWN
'Define las propiedades de la seccion
data.SetValue I_BSDV_AX, 0.00103
```

Fig. 5. Wrapper for Robot™ via its API (with a VBA® macro).

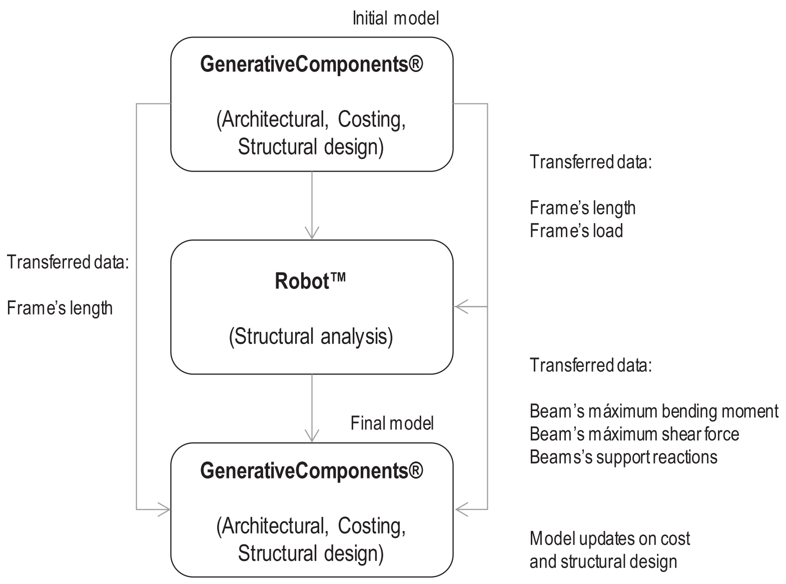

Initial model
GenerativeComponents®
(Architectural, Costing, Structural design)
Transferred data:
Frame's length
Frame's load
Transferred data:
Frame's length
Robot™
(Structural analysis)
Transferred data:
Beam's máximum bending moment
Beam's máximum shear force
Beams's support reactions
Final model
GenerativeComponents®
(Architectural, Costing, Structural design)
Model updates on cost and structural design

Fig. 6. Simulation process with selected tools to be executed within ModelCenter®.

model within Robot™ feeds the final architectural model with data from the structural analysis such as maximum bending moment, maximum shear force and beam support reactions (Table 11).

The process model worked appropriately after running in ModelCenter®. To begin, the initial file of GenerativeComponents® was opened, a change of the frame length was implemented, the file was saved and the application closed. Immediately, the updated frame length and the design load of the frame were transferred to Robot™ as input variables. These variables were introduced to the VBA® macro code from the corresponding cells in its spreadsheet. Within Robot™, a silent process was run that updated the structural frame geometry with its new length value, performed structural analysis and obtained its results. The results (maximum bending moment, maximum shear force, beam support reactions) were transferred from the macro code to their corresponding output cells in a spreadsheet, from which were extracted and transferred to a final GenerativeComponents® file. Then, this file was opened; the change of the frame length was implemented in this file as well, and the model was fed with the results of the structural analysis. Finally, the cross sections of the frame elements were internally designed in GenerativeComponents®. Therefore, every frame element was quantified and costed and the total frame direct cost was updated (Figs. 8 and 9).

The design exercise developed in this phase demonstrated that it is possible to work from PIDO with tools if they have the three previous

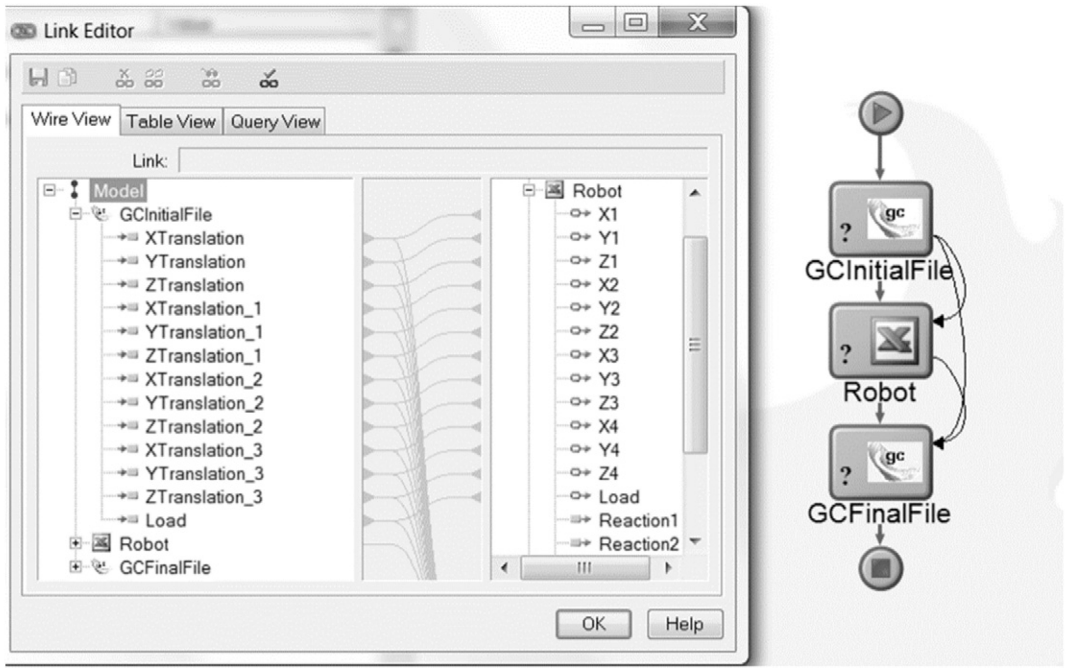


Fig. 7. Links among the tool variables to transfer data through the process model within ModelCenter®.

Table 11
Summary of the inputs and output variables to be used in the simulation process among tools.

Tool	What was modeled or performed	Inputs	Outputs	Purpose of outputs
GenerativeComponents® (initial model)	Architectural model Design of structural elements Total direct cost		Dimensions of components Frame load	Update the structural model
Robot™	Structural model Structural analysis	Dimension of components Frame load	Maximum bending moment Maximum shear force Reactions of the frame beam	Update the final architectural model
GenerativeComponents® (final model)	Architectural model Design of structural elements Total direct cost	Maximum bending moment Maximum shear force Reactions of the frame beam	Total direct cost	Update the total direct cost of the model

capabilities. However, the purpose of this simulation design exercise was not to develop MDO yet. Therefore, more design alternatives were not obtained through an optimization process in this case.

3.4. Challenges found in tools working with PIDO

There were several challenges to working with PIDO when experimenting with tools, and according to the literature, with respect to computing programming. This section first explains some programmatic challenges and considerations. Then, it summarizes the challenges found into three categories: technology, documentation and programming for each previous requisite: component interoperability, tool automation, and parameterization.

Some of the challenges related to programming were, of course, mentioned in the literature. Kelleher and Pausch [54] stated that if the user needed to customize software for personal use in order to explore ideas, such challenges would arise. In this case, the designers would need to explore non-conventional design ideas or work more efficiently. To accomplish this, designers would require computational programming in order to have better capabilities and freedom to explore [15, 55]. However, the change from working with a 3D model to work with a programming environment would introduce cognitive challenges. These cognitive challenges would be related to an understanding of the syntax of programming languages, the necessary experience, and the use of independent APIs to control the tool GUIs.

The challenges found in this research to prepare an environment for MDO included three aspects: technology, documentation, and programming. These were studied for each MDO requirement in each type of tool. Technologically, component interoperability challenges implied much experimentation and time spent studying the tool APIs and/or BP to achieve integration to PIDO. Technological automation challenges included trial and error methods to manipulate models' components and run analyses without human intervention or dealing with pop-up windows. To achieve parameterization, many hours of trial and error were required to create parameters and associations among them. Full parameterization was not achieved in BIM tools. In terms of documentation, the largest challenge was to investigate and collect all disperse information due to the lack of official documentation regarding APIs, automation and parameterization methods for BIM and PM tools. With regards to programming challenges, it was necessary to become familiar with several programming languages and tool APIs. The object-oriented programming included VB.NET®, VBA®, VBScript®, C#® and Bentley® GCScript™ according to the tool architectures. In addition, we needed to review the scope of the tool APIs, the methods, properties and behavior of their objects and, in some cases, to correct the methods' errors and bugs by programming (Tables 12, 13 and 14).

3.5. Successful strategies for working with PIDO platforms

After finishing the methodology, we detected some successful strategies that can help the design optimization researchers to save time, by taking into account some important considerations. These strategies were grouped into three categories: component Interoperability, tool automation, and parameterization. This section lists the main strategies for every requisite category.

3.5.1. Component Interoperability strategies

The strategies for component interoperability would depend on the interface utilized for tool integration. This interface would be the API or the Batch Processing. They are explained further in this subsection.

3.5.1.1. For an API. The strategies for component interoperability between a tool and a PIDO platform would depend upon the type of API and the desired degree of sophistication for the wrapper. However,

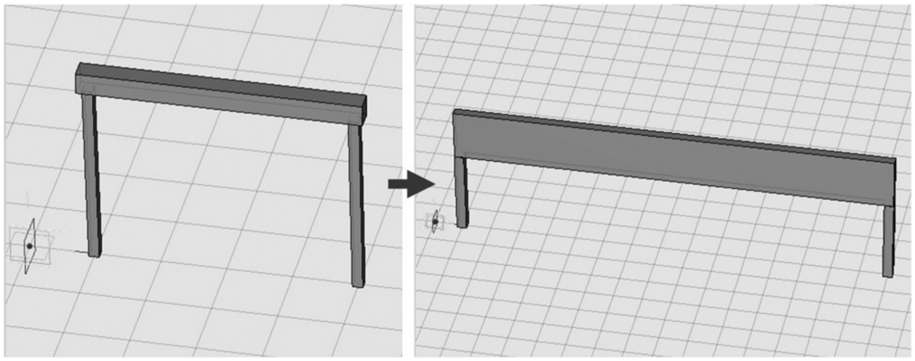
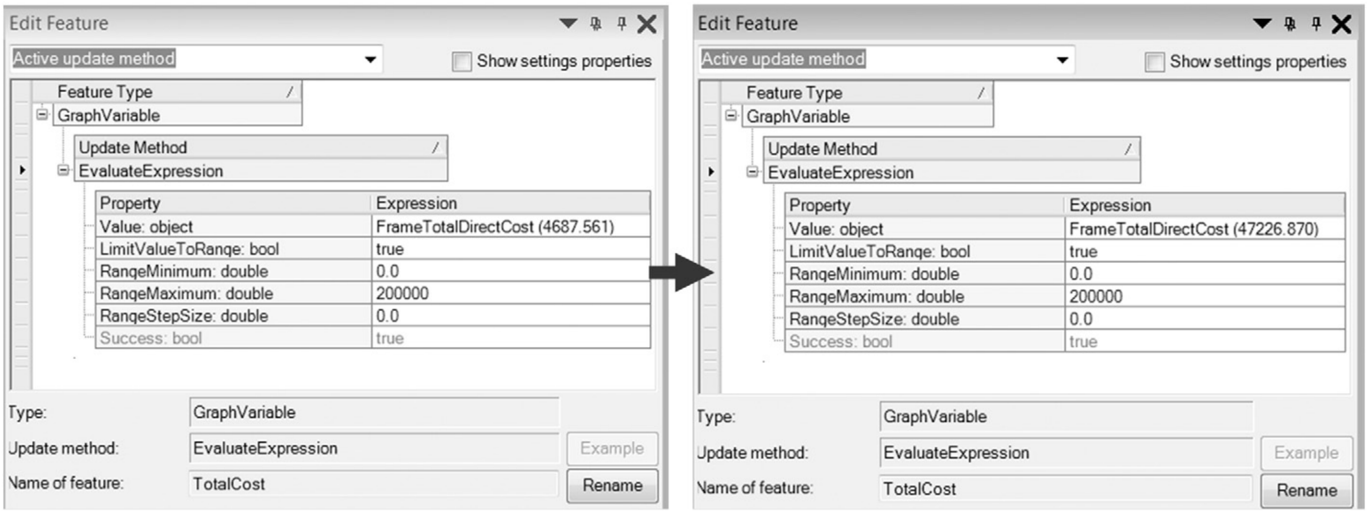


Fig. 8. The frame's length and the cross sections of the structural elements were automatically fed with Robot™ data and updated in GenerativeComponents® within ModelCenter®.



Feature Type: GraphVariable
Update Method: EvaluateExpression

Property	Expression
Value: object	FrameTotalDirectCost (4687.561)
LimitValueToRange: bool	true
RangeMinimum: double	0.0
RangeMaximum: double	200000
RangeStepSize: double	0.0
Success: bool	true

Type: GraphVariable
Update method: EvaluateExpression
Name of feature: TotalCost

Feature Type: GraphVariable
Update Method: EvaluateExpression

Property	Expression
Value: object	FrameTotalDirectCost (47226.870)
LimitValueToRange: bool	true
RangeMinimum: double	0.0
RangeMaximum: double	200000
RangeStepSize: double	0.0
Success: bool	true

Type: GraphVariable
Update method: EvaluateExpression
Name of feature: TotalCost

Fig. 9. The total direct cost of the frame was automatically updated by the simulation process in GenerativeComponents® within ModelCenter®.

some general strategies would consist of inquiring and working with the API characteristics and defining the wrapper type, by considering:

- The type of architecture and technology of the tool API. It could be object-oriented or service-oriented. Some types are COM™, CORBA®, Enterprise JavaBeans™, and Windows Communication Foundation® (WCF®)
- The possibility of controlling the tool externally through its API. This could be achieved using Excel® or another tool
- The API's degree of openness
- Documentation of classes, methods and properties of the tool APIs' objects and technical support
- Examples of object's and/or database manipulation from outside
- Type of method and functionality for the wrapper to be developed within PIDO

Some tools have an API based on COM™ objects that allows the tool to be controlled from within another application. This was the case for AECOsim Building Designer™, StaadPro® and Robot™, which were first communicated and controlled from within Excel®.

3.5.1.2. For Batch Processing. Integrating a tool with PIDO via BP was simpler than integrating it via its API, because there was no programming involved. PIDO platforms offered several methods to quickly integrate a tool. These methods included creating input and output files. Additionally, it was possible to define some types of variables such as: string, integer, double, and enumeration. It permitted representing any type of parameter in an optimization routine. However, it was important to inquire if the tool possessed the following:

- The existence of BP within the tool
- The BP capabilities and scope: printing, updating files, running commands, running macros, and so on
- BP capability to run from Windows® CLI (some BP interfaces only ran internally)
- The creation of command scripts and its syntax or the text file structure
- Documentation examples of controlling the tool from the Windows® CLI

As the first option, researchers should try this method before wrapping tools via their API because BP is much simpler than working with an API. This was the case of GenerativeComponents® that did allow writing a text file. This file was able to be run via Batch Mode.

3.5.2. Tool Automation strategies

Tool Automation could be performed by several methods depending on the tool's design and functionality. These methods were the command line interface, macros, text files and/or databases. This section

Table 12
Main challenges found in BIM tools to work with PIDO.

Challenges found in BIM tools			
	Technology	Documentation	Programming
In Component Interoperability	*Experimentation and time spent to achieve external control via the API *Experimentation and time spent when working with Batch Processing that did not work properly	*Research and collection of disperse information due to lack of official documentation on tool API (books, official forums, training and web manuals)	*Learning and experience of several object-oriented languages *Familiarity with each tool's API
In Automation	*Manipulation of specific elements in a model due to lack of preset commands for it *Completing actions due to pop-up menus that appeared when running the tools automatically	*Research and collection of disperse information on commands, chains of commands, macros and capabilities of command line because automation methods were not explicit or clear	*Familiarity and creativity with command chains and the creation of macros *Much experimentation when passing arguments to macros via the command line *Familiarity with each tool's API
In model Parameterization	Too much experimentation time trying to work with parametric associations that were not possible	Research and collection of disperse and limited information on parameterization	Familiarity with parameterization programming (for certain elements)

Table 13
Main challenges found in PM tools to work with PIDO.

Challenges found in Parametric Modeling tools			
	Technology	Documentation	Programming
In Component Interoperability	*Experimentation and time spent with Batch Processing because there were no command scripts or an externally controlled API	*Research and collection of disperse information about interfaces and alternative methods of external control	Batch Processing experience with the tool's script file and GCScript™ syntax
In Automation	*Manipulation of virtual models' specific elements due to lack of a command line interface and macros	*Research on methods to run the tool automatically	*Familiarity with script text files that recorded user actions and recreated the models
In model Parameterization	*Experimentation and time spent in the creation of custom parametric BIM content that included intelligence, behavior and specific parameters to carry out the optimization and analyses both visually and programmatically	*Research of scarce API information to create custom parametric content to allow building systems and changes of materials	*Familiarity with C#®, Script languages and the tool's API *Familiarity and training with the tool's GUI to create parametric content by visual programming

explains the first three strategic methods, since this work limited itself to their study.

When the command line method was used, tools were automated by executing them without a GUI. This execution was triggered by only pressing the enter key. Some command lines allowed combining several commands in one action, forming more complex commands. This helped when the user needed to create custom actions. For example, selecting and moving a specific wall in a model could be performed by combining the selection and move commands of the application in just one action.

However, at times, the activation of a command only triggered an initial action, and the user had to manually enter the rest of the data and choose the next command's options. In other cases, there was no command to perform a specific action in a tool. In this case, macros could be created or recorded to automate these actions in a desired sequence. Occasionally, macros were able to run and arguments were passed to them from the command line.

Text files were another method to automate a tool when there was no command line or a macro editor. These files contained all of the previously created user actions to regenerate the model. They could be modified in a text editor and then run via BP. This was the least flexible method because the recorded text files only contained the actions created by the user and, in some cases, other unrecorded actions could not be incorporated.

3.5.3. Parameterization strategies

Parameterization capabilities were crucial but were the least common in the tools. Therefore, we had to carefully consider three strategies: to select the appropriate PM tool, to become familiar with parameterization methods, and to embed behavior and analyses within the specific components of the tool. This subsection explains these general strategies.

First, the authors selected a tool that contained parametric capabilities by inquiring about the following:

- The existence and functionality of existing relationships among building elements
- The creation of custom parameters both general and component-specific
- The incorporation of parameters with complex logic
- The creation of custom associations among parameters
- The creation of custom parametric components

It was necessary to become acquainted with the tool parameterization methods, with the tool APIs, and with their programming languages, which varied from tool to tool. Familiarity with parameterization methods included four concerns: reviewing the scope of the behavior of preset components to know what would be required according to the model needs, adding parameters to the existing model's components, incorporating rules into the model's parameters and associating these rules among themselves to make the model more dynamic either by preset commands or by visual programming, and creating new components either graphically or programmatically using the APIs. This implied knowing the scope, the elements, and the programming language of the APIs.

Finally, a useful strategy was to embed multidisciplinary behavior in the parametric components themselves such as structural design and costing. Embedding part of these analyses in components enabled us to take advantage of Parametric Modeling and perform them internally and quickly rather than needing to redirect the information and perform them within third-party applications. Third-party applications were able to complete the other part of the analyses in an optimization routine. For example, when performing structural analysis and structural design, the calculation of components' internal forces was performed by a structural third-party application and the rules to design the structural components were programmed within the components themselves.

4. Conclusions

This research provides a solid foundation to help researchers develop MDO by integrating applications to PIDO platforms in order to create a design and optimization workflow with the tools involved. According to our findings, this development involves four issues. First, the evaluation of computing applications in order to understand their behavior, advantages and disadvantages regarding component interoperability,

Table 14
Main challenges found in structural analysis tools to work with PIDO.

Challenges found in Structural tools			
	Technology	Documentation	Programming
In Component Interoperability	Experimentation and time spent when interfacing the tools with PIDO		*Familiarity with tool API
In Automation	Experimentation and time spent when automating every step of modeling and structural analysis processes to validate the results programmatically		*Familiarity with tool commands and special functions
In model Parameterization	Experimentation and time spent when parameterizing some variables to validate the results programmatically		

automation of tools, and parameterization of models. Second, knowledge of the possible challenges with respect to technology, lack of documentation, and programming issues. Third, the awareness of strategies that can be implemented to help develop MDO. And finally, the simulation of an automatic design process that involved two applications integrated to a PIDO platform.

Our hypothesis is false, since not any tool can be totally used with a PIDO platform. These tools have been principally designed to be operated by the graphical user interface. Although they are very useful tools in the AEC industry, their developers have not fully included capabilities in their design regarding the three requirements described: component interoperability, automation and parameterization. For example, the majority of them caused concerns principally with respect to automation and parameterization capabilities. Regarding component interoperability, the main issues detected were a low openness level of the tool interfaces that did not enable a full integration with PIDO nor permit access to a models' properties, as well as the high computational time to control the tools via Batch Processing. In automation, the scarcity of available preset commands to perform tasks impeded manipulation of analyses and modification of models. Another problem was the presence of pop-up menus enquiring user data entries. Lastly, the tools showed problems in parameterization capabilities because the majority did not permit the creation of real parametric associations among components, the compatibility among certain existing components, or the creation of parametric components, which are limited in the current BIM and PM tools.

The difficulty to parameterize was mainly due to two things: the way classes were encapsulated and the way Parametric Modeling algorithms were implemented in modeling tools. Regarding encapsulation, although all the modeling tools tested were object oriented, their objects had different behavior since their classes were encapsulated differently. For example, in Revit, system components (walls, slabs, beams and so on) are instanced by public classes whose constructor methods are internal, and that contain private methods and properties, which cannot be accessed programmatically. Additionally, classes are limited only to the assembly in which they were declared. This is also why new custom system families in Revit cannot be created, it is only possible to access to certain properties and methods like the “duplicate” method in Revit families. However, unlike Revit® or AECOsim Building Designer™, GenerativeComponents® contains public classes with public methods and properties to create custom parametric content programmatically. Regarding Parametric Modeling, it is based on three types of algorithms: a node-sort algorithm, a propagation algorithm and a graph algorithm. The functionality of these algorithms is quite explicit in PM tools such as GenerativeComponents®. However, although the BIM tools are based on Parametric Modeling, these algorithms work on a preset and internal way to automatically sort BIM components and propagate changes according to their predesigned relationships, which are not displayed. Therefore, these parametric relationships cannot be known or edited. GenerativeComponents®, as opposed to Revit or AECOsim Building Designer™, allows the user to create custom parametric relationships providing flexibility in the model.

However, the increasing use of MDO through PIDO platforms in the AEC industry will require tools that can interact in a single process. To achieve this, tools should include additional capabilities with respect to the three requirements already described. There is an area of opportunity for tool developers to improve future tool capabilities.

Challenges can be expected when developing component interoperability, automation and parameterization. This research grouped these changes into three categories: technology, documentation, and programming. The main technological difficulties were time consuming experimentation needed to couple each tool with PIDO, automate their processes and parameterize their models. Other principal difficulties were working with limited official documentation on the tools' non-graphical interfaces, commands, automation, and parameterization methods as well as to become familiar with each tool's API and their script programming languages.

To address the tools' identified behavior, we suggest some general strategies. For component interoperability, seek tools whose interfaces are open. This provides sufficient freedom to integrate tools into PIDO by controlling them from within third-party applications or from Windows® CLI, in the case of Batch Processing. For automation, test and validate sequentially that each action or process flows without user intervention through the tool's GUI so that these processes finish and generate outputs. Lastly, for parameterization, test the creation of parameters, relationships among them, and parametric components with embedded multidisciplinary behavior.

The design process created within PIDO in phase three worked appropriately. After the tools were wrapped, it was possible to create a design process model within PIDO and perform simulations in only one environment. This simulation automatically controlled the tools in the desired sequence. Data were transferred from one tool to another successfully, producing the correct outputs for the frame example, in terms of the frame's dimensions, sections of structural elements, and direct cost. This exercise demonstrated that it is possible to work from PIDO if the tools have the three previous capabilities. However, the purpose of this simulation was not to develop MDO for this article. MDO exercises for the production of multiple design alternatives dealing with multiple conflicting objectives will be addressed in future research.

Because the tools are still undeveloped, developing MDO continues to be a research topic. For future work, authors should test new tools by interfacing them with optimization environments such as PIDO. These methods would need to include integrating tools via Windows Communication Foundation®.

On the other hand, PM tools can be useful to automatically explore design space from conceptual design to overcome some current limitations of MDO. Further research is necessary to formalize part of the complex design process.

Acknowledgements

We gratefully acknowledge Instituto Tecnológico y de Estudios Superiores de Monterrey and Pontificia Universidad Católica de Chile who provided all the academic software licenses used in this research. Additionally, we acknowledge Dr. Socorro Marcos, Dr. Celia Ann Durboraw and the reviewers of this journal for their invaluable technical revision of this work. Finally, we acknowledge to CONICYT and CONACYT for their economical support of this research.

References

[1] W. Wang, H. Rivard, R. Zmeureanu, An object-oriented framework for simulation-based green building design optimization with genetic algorithms, Advanced Engineering Informatics. 19 (2005) 5–23, http://dx.doi.org/10.1016/j.aei.2005.03.002.
[2] J.K. Liker, D. Meier, The Toyota Way Fieldbook: A Practical Guide for Implementing Toyota's 4Ps, 1st ed McGraw-Hill Companies, United States of America, 2006 (ISBN: 978-0071448932).
[3] F. Flager, G. Soremekun, B. Welle, J. Haymaker, P. Bansal, Multidisciplinary process integration and design optimization of a classroom building, Electron. J. Inf. Technol. Constr. 14 (2009) 595–612http://www.itcon.org/2009/38.
[4] Z. Ren, F. Yang, N.M. Bouchlaghem, C.J. Anumba, Multi-disciplinary collaborative building design—a comparative study between multi-agent systems and multi-disciplinary optimisation approaches, Automation in Construction. 20 (2011) 537–549, http://dx.doi.org/10.1016/j.autcon.2010.11.020.
[5] Y. Fan, D. Bouchlaghem, Genetic Algorithm-Based Multiobjective Optimization for Building design, Arc. Eng. Des. Manage. 6 (2010) 68–82, http://dx.doi.org/10.3763/aedm.2008.0077.
[6] M. Fragiadakis, N.D. Lagaros, An overview to structural seismic design optimisation frameworks, Computers & Structures. 89 (2011) 1155–1165, http://dx.doi.org/10.1016/j.compstruc.2010.10.021.
[7] C. Chen, M. Usman, Design optimisation for automotive applications, International Journal of Vehicle Design. 25 (2001) 126–141, http://dx.doi.org/10.1504/IJVD.2001.001912.
[8] J. Sobieszczanski-Sobieski, R.T. Haftka, Multidisciplinary aerospace design optimization: survey of recent developments, Struct. Opt. 14 (1997) 1–23, http://dx.doi.org/10.1007/BF01197554.

[9] M. D'Auria, R. D'Ippolito, Process integration and design optimization ontologies for next generation engineering, in: Y. Tang, H. Panetto (Eds.), On the Move to Meaningful Internet Systems: OTM 2013 Workshops, Springer, Berlin Heidelberg 2013, pp. 228–237 (ISBN: 978-3-642-41032-1).
[10] R. Morales-Rodríguez, R. Gani, S. Déchelotte, A. Vacher, O. Baudouin, Use of CAPE-OPEN standards in the interoperability between modelling tools (MoT) and process simulators (Simulis® Thermodynamics and ProSimPlus), Chemical Engineering Research and Design. 86 (2008) 823–833, http://dx.doi.org/10.1016/j.cherd.2008.02.022.
[11] M. Groover, E. Zimmers, CAD/CAM: Computer-Aided Design and Manufacturing, 1st ed Prentice Hall, United States of America, 1984 (ISBN: 978-0131101302).
[12] C. Eastman, P. Teicholz, R. Sacks, K. Liston, BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors, 2nd ed. Wiley, 2011 (ISBN: 978-0470541371).
[13] W. Kymmell, Building Information Modeling: Planning and Managing Construction Projects with 4D CAD and Simulations, 1st ed. United States of America, McGraw-Hill Education, 2008 (ISBN: 978-0071494533).
[14] C. Derix, In-Between Architecture Computation, International Journal of Architectural Computing. 7 (2009) 565–586, http://dx.doi.org/10.1260/1478-0771.7.4.565.
[15] R. Woodbury, Elements of Parametric Design, 1st ed Routledge, USA and Canada, 2010 (ISBN: 978-0-415-77987-6).
[16] B. Welle, J. Haymaker, Z. Rogers, ThermalOpt: A methodology for automated BIM-based multidisciplinary thermal simulation for use in optimization environments, Building Simulation. 4 (2011) 293–313, http://dx.doi.org/10.1007/s12273-011-0052-5.
[17] E. Mark, Optimizing solar insolation in transformable fabric architecture: a parametric search design process, Automation in Construction. 22 (2012) 2–11, http://dx.doi.org/10.1016/j.autcon.2011.11.001.
[18] D. Tuhus-Dubrow, M. Krarti, Genetic-algorithm based approach to optimize building envelope design for residential buildings, Building and Environment. 45 (2010) 1574–1581, http://dx.doi.org/10.1016/j.buildenv.2010.01.005.
[19] V. Granadeiro, J.P. Duarte, J.R. Correia, V.M.S. Leal, Building envelope shape design in early stages of the design process: Integrating architectural design systems and energy simulation, Automation in Construction. 32 (2013) 196–209, http://dx.doi.org/10.1016/j.autcon.2012.12.003.
[20] X. Shi, W. Yang, Performance-driven architectural design and optimization technique from a perspective of architects, Automation in Construction. 32 (2013) 125–135, http://dx.doi.org/10.1016/j.autcon.2013.01.015.
[21] J. Plume, J. Mitchell, Collaborative design using a shared IFC building model—Learning from experience, Automation in Construction. 16 (2007) 28–36, http://dx.doi.org/10.1016/j.autcon.2005.10.003.
[22] R. Sacks, I. Kaner, C.M. Eastman, Y. Jeong, The Rosewood experiment — Building information modeling and interoperability for architectural precast facades, Automation in Construction. 19 (2010) 419–432, http://dx.doi.org/10.1016/j.autcon.2009.11.012.
[23] J. Steel, R. Drogemuller, B. Toth, Model interoperability in building information modelling, Software & Systems Modeling. 11 (2012) 99–109, http://dx.doi.org/10.1007/s10270-010-0178-4.
[24] M. Marzouk, A. Abubakr, Decision support for tower crane selection with building information models and genetic algorithms, Automation in Construction. 61 (2016) 1–15, http://dx.doi.org/10.1016/j.autcon.2015.09.008.
[25] I. Abotaleb, K. Nassar, O. Hosny, Layout optimization of construction site facilities with dynamic freeform geometric representations, Automation in Construction. 66 (2016) 15–28, http://dx.doi.org/10.1016/j.autcon.2016.02.007.
[26] V. Faghihi, K.F. Reinschmidt, J.H. Kang, Objective-driven and Pareto Front analysis: Optimizing time, cost, and job-site movements, Automation in Construction. 69 (2016) 79–88, http://dx.doi.org/10.1016/j.autcon.2016.06.003.
[27] I.G. Dino, An evolutionary approach for 3D architectural space layout design exploration, Automation in Construction. 69 (2016) 131–150, http://dx.doi.org/10.1016/j.autcon.2016.05.020.
[28] S. Chardon, B. Brangeon, E. Bozonnet, C. Inard, Construction cost and energy performance of single family houses: From integrated design to automated optimization, Automation in Construction. 70 (2016) 1–13, http://dx.doi.org/10.1016/j.autcon.2016.06.011.
[29] M. Ferrara, E. Fabrizio, J. Virgone, M. Filippi, Energy systems in cost-optimized design of nearly zero-energy buildings, Autom. Constr. 70 (2016) 109–127 (doi: j.autcon.2016.06.007).
[30] J.P. Basbagill, F.L. Flager, M. Lepech, A multi-objective feedback approach for evaluating sequential conceptual building design decisions, Automation in Construction. 45 (2014) 136–150, http://dx.doi.org/10.1016/j.autcon.2014.04.015.
[31] J. van Hellenberg Hubar, Design concept for optimizing the renewable micro generation technologies to supply an off-grid community energy demand: A case study with simulation model in the Netherlands, Eindhoven University of Technology, 2011.
[32] X. Shi, Design optimization of insulation usage and space conditioning load using energy simulation and genetic algorithm, Energy 36 (2011) 1659–1667, http://dx.doi.org/10.1016/j.energy.2010.12.064.
[33] B. Lee, M. Trcka, J.L.M. Hensen, Rooftop photovoltaic (PV) systems for industrial halls: Achieving economic benefit vial lowering energy demand, Front. Architec. Res. 1 (2012) 326–333, http://dx.doi.org/10.1016/j.foar.2012.09.003.
[34] A. Chronis, K.A. Liapi, I. Sibetheros, A parametric approach to the bioclimatic design of large scale projects: The case of a student housing complex, Automation in Construction. 22 (2012) 24–35, http://dx.doi.org/10.1016/j.autcon.2011.09.007.
[35] M. Turrin, P. von Buelow, A. Kilian, R. Stouffs, Performative skins for passive climatic comfort. A parametric design process, Automation in Construction. 22 (2012) 36–50, http://dx.doi.org/10.1016/j.autcon.2011.08.001.
[36] A. Albertin, P.G. Malerba, N. Pollini, M. Quagliaroli, Prestress optimization of hybrid tensile structures, in: F. Biondini, D.M. Frangopol (Eds.),Bridge Maintenance, Safety, Management, Resilience and Sustainability: Proceedings of the Sixth International IABMAS Conference 2012, pp. 1750–1757, http://dx.doi.org/10.1201/b12352-256.
[37] P. Geyer, M. Buchholz, Parametric systems modeling for sustainable energy and resource flows in buildings and their urban environment, Automation in Construction. 22 (2012) 70–80, http://dx.doi.org/10.1016/j.autcon.2011.07.002.
[38] S. Attia, M. Hamdy, W. O'Brien, S. Carlucci, Assessing gaps and needs for integrating building performance optimization tools in net zero energy buildings design, Energy Build. 60 (2013) 110–124, http://dx.doi.org/10.1016/j.enbuild.2013.01.016.
[39] V. Granadeiro, L. Pina, J.P. Duarte, J.R. Correia, V.M.S. Leal, A general indirect representation for optimization of generative design systems by genetic algorithms: Application to a shape grammar-based design system, Automation in Construction. 35 (2013) 374–382, http://dx.doi.org/10.1016/j.autcon.2013.05.012.
[40] J.N. Richardson, G. Nordenson, R. Laberenne, R. Filomeno Coelho, S. Adriaenssens, Flexible optimum design of a bracing system for façade design using multiobjective Genetic Algorithms, Automation in Construction. 32 (2013) 80–87, http://dx.doi.org/10.1016/j.autcon.2012.12.018.
[41] F. Flager, D.J. Gerber, B. Kallman, Measuring the impact of scale and coupling on solution quality for building design problems, Design Studies. 35 (2014) 180–199, http://dx.doi.org/10.1016/j.destud.2013.11.001.
[42] S.E. Lin, D.J. Gerber, Designing-in performance: A framework for evolutionary energy performance feedback in early stage design, Automation in Construction. 38 (2014) 59–73, http://dx.doi.org/10.1016/j.autcon.2013.10.007.
[43] A. Karatas, K. El-Rayes, Optimizing tradeoffs among housing sustainability objectives, Automation in Construction. 53 (2015) 83–94, http://dx.doi.org/10.1016/j.autcon.2015.02.010.
[44] C.T. Mueller, J.A. Ochsendorf, Combining structural performance and designer preferences in evolutionary design space exploration, Automation in Construction. 52 (2015) 70–82, http://dx.doi.org/10.1016/j.autcon.2015.02.011.
[45] K. Negendahl, Building performance simulation in the early design stage: An introduction to integrated dynamic models, Automation in Construction. 54 (2015) 39–53, http://dx.doi.org/10.1016/j.autcon.2015.03.002.
[46] M. Madiajagan, B. Vijayakumar, Interoperability in Component Based Software Development, International Journal of Computer, Electrical, Automation, Cont. Info. Eng. 2 (2008) 3507–3515http://www.waset.org/publications/10105.
[47] V. Issarny, M. Caporuscio, N. Georgantas, A perspective on the Future of Middleware-based Software Engineering, Future Software Eng. 2007 (2007) 244–258, http://dx.doi.org/10.1109/FOSE.2007.2.
[48] W. Emmerich, in: A. Finkelstein (Ed.), Software engineering and middleware: a roadmapProceedings of the Conference on The Future of Software (ICSE '00) 2000, pp. 117–129, http://dx.doi.org/10.1145/336512.336542.
[49] J. Oberleitner, T. Gschwind, M. Jazayeri, in: H. Hart (Ed.), The Vienna Component Framework enabling composition across component modelsProceedings of the 25th International Conference on Software Engineering (ICSE '03) 2003, pp. 25–35, http://dx.doi.org/10.1109/ICSE.2003.1201185.
[50] G. Eddon, H. Eddon, Inside Distributed COM, Microsoft Press, Redmond, Washington, 1998 (ISBN:1-57231-849-X).
[51] M. Henning, S. Vinoski, Advanced CORBA Programming with C++, 1st ed Addison Wesley Longman, Inc., United States of America, 2008 (ISBN: 978-0201379273).
[52] E. Roman, R.P. Sriganesh, G. Brose, Mastering Enterprise JavaBeans, Third ed Wiley, United States of America, 2005 (ISBN: 978-0764576829).
[53] J. Koller, A. Kuckelberg, CAPE-OPEN (CO) standards: implementation and maintenance, Standardization and Innovation in Information Technology, 2001 2nd IEEE Conference 2001, pp. 335–338, http://dx.doi.org/10.1109/SIIT.2001.968582.
[54] C. Kelleher, R. Pausch, Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers, ACM Computing Surveys. 37 (2005) 83–137, http://dx.doi.org/10.1145/1089733.1089734.
[55] R. Aish, Extensible computational design tools for exploratory architecture, in: B. Kolarevic (Ed.), Architecture in the Digital Age: Design and Manufacturing, Spon Press, New York, NY 2003, pp. 243–252 (ISBN:9780203634561).