Research in Engineering Design (2020) 31:25–52
https://doi.org/10.1007/s00163-019-00321-9

ORIGINAL PAPER

A modelling framework to support design of complex engineering systems in early design stages

Shiva Abdoli[1] · Sami Kara[1]

Received: 4 October 2017 / Revised: 26 June 2019 / Accepted: 26 August 2019 / Published online: 12 September 2019

Abstract
Production, assembly or logistic systems exist in widespread domains. It is agreed that more than 50% of life-cycle performance, costs and environmental impacts of such systems are due to those decisions that are made in their early design stages (Reich, Res Eng Design 28(4):411–419, https://doi.org/10.1007/s00163-017-0270-7, 2017). However, the large scale and multi-disciplinary essence of such systems make their design considerably challenging. Most of the design approaches follow a sequential approach such that the design in each lower level is finalized/frozen before proceeding to the next level. However, such approaches do not properly address the interaction between different design disciplines which may later lead to design inconsistencies. Therefore, this paper aimed to propose a modelling framework that allows having an integrated approach in the early design stages of such systems. To this end, first the framework prescribed developing an executable meta-architecture that can embody all the design requirements. Second, the framework describes the interconnections between the meta-architecture with certain supporting algorithms and optimization models. This allows generating and simulating different design alternatives and observing the impact of different design decisions on system integrated performance. Therefore, the proposed framework with its providing outcomes can be used to support the decision making in early design stages of such systems. The framework is applied in a real case study from the warehousing domain, which serves to show the practical application of the proposed framework.

Keywords Complex engineering systems · Object Oriented modelling · Systems engineering · System logical architecture · Discrete event simulation · Finite state machine

1 Introduction

1.1 Background

Manufacturing, assembly, and logistics systems are the main elements in supply chains of many industries, such as agriculture, fabrication, and electronics. The general performance characteristics of aforementioned systems are dependent on the decisions that are made in their initial design stages (Gu et al. 2001; Christophe et al. 2010; Umeda et al. 1996; Reich 2017). Different references treated the targeted systems as complex systems from the design perspective (Koo 2005; Bar-Yam 2002; Schuh et al. 2008; Wolfram 1985; ElMaraghy et al. 2005; Vrabič and Butala 2011). Although it is hard to find a globally accepted definition for complex systems, it is agreed that complex systems usually comprise of a large number of interacting elements and studying complex systems requires research across multiple disciplines (Wolfram 1985; Bar-Yam 2002; Schuh et al. 2008).

The targeted systems operate based on certain engineering processes. For instance, a warehouse system includes several handling, sorting, packaging, and unpacking processes. However, internal operations of processes are independent and each may perform several operations. Thus, being a large scale is one of the main characteristics of the

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s00163-019-00321-9) contains supplementary material, which is available to authorized users.

✉ Shiva Abdoli
s.abdoli@student.unsw.edu.au

Sami Kara
S.kara@unsw.edu.au

1 Sustainability in Manufacturing and Life Cycle Engineering Research Group, School of Mechanical and Manufacturing Engineering, University of New South Wales, Sydney, Australia

targeted systems. The processes dynamically interact to fulfil the overall system functionality, particularly to transit the items (e.g. products) through the system. The overall system behaviour in terms of item transition can be characterized by some events happening at discrete time points (McGinnis et al. 2006; Mönch et al. 2011). Therefore, being dynamic is another characteristic of the targeted systems. Each process may operate using diverse types of resources, including soft and physical. These systems accomplish their functionality by proper integration of both soft and hard elements. Hence, the design of such systems requires the involvement of multiple design disciplines (Zheng et al. 2016; Gausemeier et al. 2011). Therefore, being multidisciplinary is another characteristic of such systems. Emergent behaviour is a key aspect of complex systems, which happens due to the interactions between system elements. However, it is not easy to precisely predict the emergent behaviour prior to observing the dynamic interactions (White 2007; Bar-Yam 2004). The combination of the aforementioned characteristics lends these systems to the complex system class from the design perspective (Koo 2005; Bar-Yam 2002; Schuh et al. 2008; Wolfram 1985; ElMaraghy et al. 2005; Braha et al. 2006; Vrabič and Butala 2011; McGinnis et al. 2006, 2011). Hence, the targeted systems are called 'Complex Engineering Systems' (CESs), which is an umbrella term referring to the aforementioned systems in this paper.

1.2 Research scope

This paper aims to propose a framework to support the CES design, at the early design stages, in three ways as given below. The motivations for the following objectives are described afterwards.

1. Managing the complexity of the design knowledge by providing a prescriptive approach for structuring the design knowledge.
2. Enabling integrated design by having a holistic design approach in terms of addressing both non-physical (operational policies) and physical aspects of the system concurrently. The holistic approach should allow observing the impact of design decisions on the dynamic behaviour of the system.
3. Enabling integrated (multidisciplinary) design to facilitate achieving design consistency.

The design knowledge includes a broad range of concepts, including objectives, constraints, and requirements. Managing the complexity of the design knowledge is a challenging task in CES design (ElMaraghy et al. 2005; INCOSE 2015; Chakrabarti and Blessing 2014; Sitton and Reich 2018). Particularly, the manner of structuring the design knowledge in a proper artefact (model) has a direct impact on realizing the design integration (Chakrabarti and Blessing 2014). Such complexity usually makes the designers from different disciplines to design in silos (Fishwick 2007; Baker and Canessa 2009; Rouwenhorst et al. 2000; Mönch et al. 2011). Existing design approaches mostly follow a sequential approach, which hardly integrates the various design stages (Cochran et al. 2000). Therefore, the connection between low-level design decisions and high-level system objectives cannot be captured properly, so it is not easy to recognize how design decisions at various stages affect the overall system performance. Moreover, it is likely that some aspects of the system and its design knowledge are being overlooked in such sequential approaches (Tomiyama et al. 2007). Hence, having a holistic approach is essential to realize integrated-multidisciplinary design (Maropoulos and Ceglarek 2010), otherwise inconsistency between different design disciplines and failures may happen. Particularly, operational policies or non-physical aspects of CESs are mostly designed when the physical design is finalized (frozen). For example in designing a warehouse, the designers may prefer to design narrow aisles for saving on space cost. Yet, double-command-storing-picking (storing and picking in one aisle simultaneously) is not feasible (consistent) with narrow-aisles. Design failures because of such inconsistencies happen quite often when designers from different disciplines make their decisions separately (Komoto and Tomiyama 2012). This emphasizes the importance of proper structuring of the design knowledge in an artefact that allows an integrated design to realize the design consistency. The sequential approach may lead to shrinking the solution space and losing a better solution (Wang 2012; Pape et al. 2013). Designing the warehouse with 'wide aisles' in conjunction with double-command mode not only can improve the throughput, but may reduce the required numbers of pickers, which results in a lower total cost. However, a systematic integration between different design disciplines is rarely addressed in the current literature (Zheng et al. 2016).

Generally, CESs achieve their intended overall functionality through the proper dynamic interaction of their processes. Although simulation is a promising approach for observing the dynamic behaviour of CESs, it is often used relatively late in the design process when the important design decisions are already taken and key performance criteria are determined. Hence, it is crucial to observe the impact of the single design decisions (in different disciplines) on the dynamic behaviour of the system, so designers can make more realistic decisions at early design stages.

To the best of the authors' knowledge, there is a need for a modelling framework that can assist in integrated design of CESs (McGinnis et al. 2006; Thiers 2014; Wang and Dagli 2008; Tomiyama et al. 2009; Mönch et al. 2011; Zheng et al. 2016). The framework should allow integrating the design in different disciplines in a holistic manner, such that the design ‘freeze’ happens on a multidisciplinary ground.

2 Literature review

Modelling in early design stages is defined as the transition from a problem situation, model requirements to a definition of what is going to be modelled and how (Robinson 2006). In the CES design context, system architecture should embody the requirements in a model that demonstrates the system’s desired function (Dori 2002; Abdoli and Kara 2017b).

Wagenhals et al. (2003) proposed a modelling approach for system architecting of systems that are related to Department of Defence missions. Application of object oriented (OO) approach using Unified Modelling Language (UML) was recommended to model the system architecture. The framework suggested transforming the UML model to Coloured Petri Net (CPN) formalism to provide an executable architecture (model to model: MtM). Ultimately, the executable model was used to simulate/observe the behavioural performance aspects of a design alternative.

In another work, a modelling framework was introduced by the application of OPM (Object Process Methodology), CPN, and ‘feature model’(Wang and Dagli 2013). Feature model was used to visualize the system elements and their relationships. OPM model was used as a hub between CPN and feature model. CPN formalism was used to simulate the dynamic behaviour of the alternatives.

Thiers introduced a design methodology to support the analysis of logistics systems by automating the process of building the analysis models (simulation) from a descriptive model of the system architecture (Thiers 2014). System modelling language (SysML) and CPN were used, respectively, to present the descriptive model of the architecture and to make the executable model. It was suggested using an automated builder program to make a CPN model from a SysML model.

Meng introduced an approach to model a reconfigurable manufacturing system by application of coloured timed object-oriented Petri Nets (PN) (Meng 2010). Material flow characteristics and time constraints were modelled as tokens attributes. However, the quantitative formulation of the problem to achieve a rigorous system configuration was not addressed.

These references (Wang 2012; Wagenhals et al. 2003; Meng 2010; Thiers 2014) mainly recommended a modelling approach and relied on designer’s tacit knowledge for system architecting (Alves and Silva 2009). However, a prescriptive modelling approach reduces the required effort for modelling the design knowledge and supports the design processes more effectively (Schotborgh et al. 2012).

Koo developed object-process network (OPN) as a modelling language, for system architecting based on OO approach and PN formalism (Koo 2005). OPN has flaws in providing explicit mechanisms for automatically generating alternatives and modelling the constraints that are related to the integration of different objects. Moreover, OPN has shortcomings regarding representing static relationships between system entities.

Dauby and Dagli (2011) proposed a methodology for the assessment of the system alternatives (Dauby and Dagli 2011). The ‘extensible modelling’ was defined as modelling the system attributes with a hierarchical structure, which each higher level aggregated the lower level parameters. ‘Canonical Design Primitives’ were defined as basic representations for genres of the system’s physical components. ‘Sensitivity functions’ were defined to predict the impact of trading one design primitive with another. It was assumed that the system architecture and mathematical relationships for sensitivity functions are known. Moreover, the impact of each design primitive on the alternative performance was assessed in isolation without taking into account the possible interactions between system elements.

Ziv-Av and Reich (2005) introduced a hierarchical approach for product-concept generation from the given customer requirements (Ziv-Av and Reich 2005). The highest layer addressed the requirements and the following levels can address their possible relationships. This work used a quadratic programming approach to formulate a solution and assess objectives satisfaction. The presentation of the requirements was similar to the house of quality development. However, the presented approach addressed the product concept development according to its physical components and studying the operational procedures of a CES received less attention. Moreover, analysing the dynamic behaviour of a product or system was not addressed.

Most of the existing frameworks basically describe the system architecture with a static language, such as UML or SysML, which are powerful in visualizing the architecture from a specific point of view. Yet, such languages do not provide substantial execution semantics compared to PNs for capturing the dynamic behaviour of the system (simulation). Although PN (or CPN) formalism has promising features for computational analysis, it has shortcomings in visualizing the system architecture meaningfully and simply (Khan 2010; Jørgensen 2004; Wagenhals et al. 2003; Yaroker et al. 2013; Wang 2012). In the current literature, it is tried to bridge this gap using a complex chain of MtM transformation (Fleck et al. 2013; Bortolini et al. 2016). Yet, developing and working with MtM transformation-based approaches require a high level of expertise in programming. On the other hand, such transformation models do not completely exist and are still evolving (Matei and Bock 2012; Fleck et al. 2013; Alvarez Cabrera et al. 2010; Moses 2002;

Mijatov et al. 2015; Mayerhofer et al. 2013; Kapos et al. 2014; Zhow et al. 2015; Nikolaidou et al. 2012; Huang et al. 2007). OPM can be located between PN and UML/SysML. Although OPM visualizes the system architecture meaningfully, it does not include a well-established computational and execution semantics compared to PN. For instance, this reference (Dori et al. 2016) used MtM transformation between OPM and Matlab to enhance the OPM capabilities by its integration with Matlab computational capabilities. More issues regarding the MtM transformation are discussed comprehensively in Kapos (2015).

Sitton and Reich (2018) introduced a framework for coordination between the systems in an enterprise (Sitton and Reich 2018). The introduced approach mainly focuses on improving the core processes that define the enterprise. Hence, the paper addresses the architecture development when the enterprise systems already exist. Therefore, the scope of the paper is different from the design at the early stages. Hence, model-based simulation has not received its deserved attention.

The papers that used MtM (Wang 2012; Wagenhals et al. 2003; Meng 2010; Thiers 2014; Schönherr and Rose 2009; McGinnis 2006 and Ustun 2009) mostly modelled the operational policies as an invariant part of the model structure. Hence, simulation of different alternatives did not address different operational procedures. However, the decisions regarding the selection of operational policies can strongly affect the dynamic behaviour of a system. For example, Thiers (2014) suggested modelling the system procedures in behavioural diagrams of SysML, which dictates the operational procedures to the generated alternatives. Similarly, the developed model by reference (Wang 2012) could not generate alternatives that vary in their operational policies. Therefore, existing frameworks have flaws in their ability to capture the design knowledge holistically in terms of addressing both structural and behavioural aspects in one model (Abdoli and Kara 2017a).

As conclusion, four specific shortcomings are identified in the reviewed frameworks, as summarized;

1. Lack of prescriptive guidelines for system architecting.
2. Demonstrating the system architecture and realizing the dynamic behaviour in separated modelling formalisms.
3. Modelling operational policies as the model invariants.
4. Lack of an integrated modelling framework with the following characteristics: first, addresses different design aspects including; generation of different alternatives and their assessments, and second, provides a systematic interconnection between different models such that they can interchange their results.

3 Modelling framework

3.1 Framework overall structure

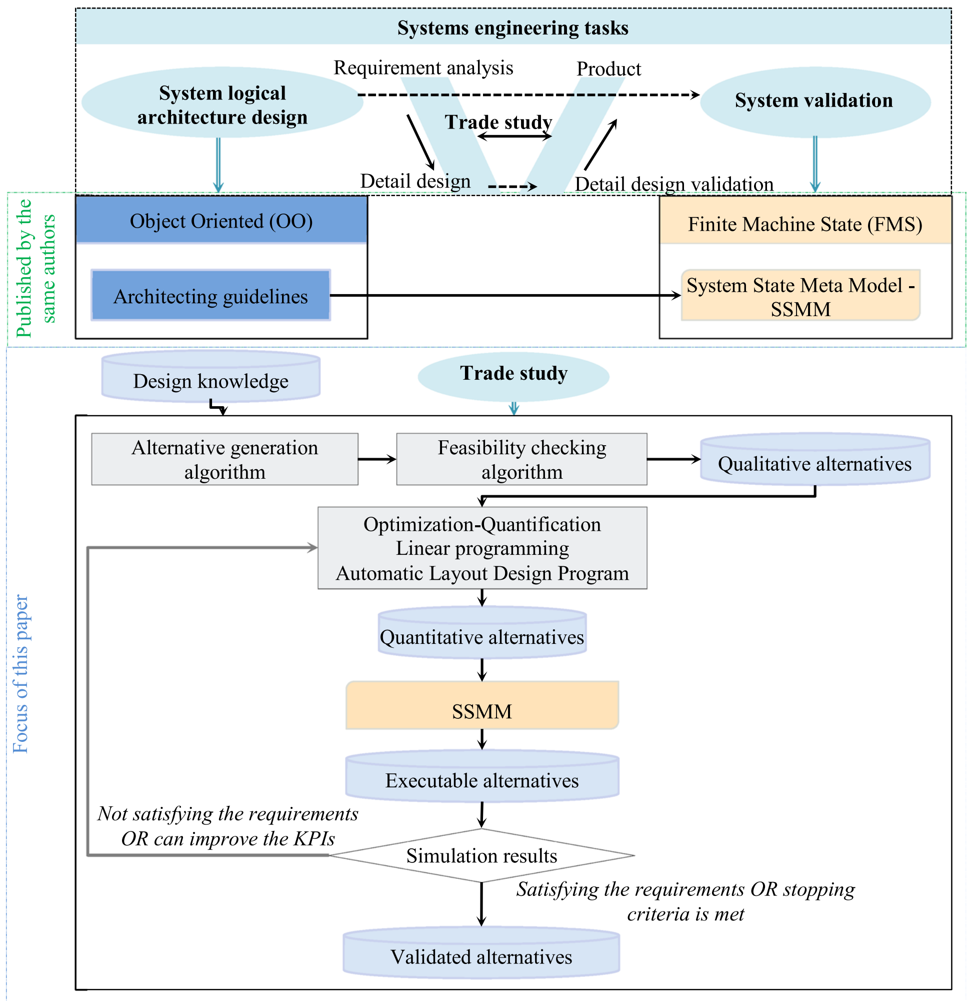
A prescriptive modelling framework clarifies the design tasks, their logical sequence, and how to perform those tasks (Albers and Braun 2011; Tomiyama et al. 2009; Estefan 2003; Cloutier and Verma 2007). Systems Engineering (SE) is a well-known approach to assist with the design of multidisciplinary systems and concentrates on system properties (Kossiakoff et al. 2011). Hence, this paper uses SE principles to define the overall structure of the proposed framework in terms of the needed tasks and their logical sequence.

From the design perspective, the SE process can be broken into four fundamental tasks, as shown in the upper section of Fig. 1. After the requirement analysis, the system engineer designs the system's logical architecture in terms of system decomposition to logical subsystems, developing their interaction interface, and allocating the requirements to them (Douglass 2016; Osorio et al. 2011). Design at early stages is considered equivalent to designing the system's logical architecture (Komoto and Tomiyama 2012). Different design alternatives are configured by selecting from possible key options for design requirements without focusing on their detail design. The goodness of the design alternatives can be assessed with respect to certain measures of effectiveness (MoEs) and accordingly a preferred alternative is nominated for the detail design, this analysis is called trade study. Down-stream engineers develop the detailed designs of subsystems. A design alternative is needed to be validated to assure its conformance to the requirements.

In the V-model of SE, the activities on the left side define what is to be designed, building a foundation for detail design. The right-side activities focus on integration-validation, trying to ensure that the development process delivers an outcome that conforms to requirements. V-model gives great attention to the integration-validation activities from the beginning of the development process, so it tries to directly interconnect the integration-validation phase with system logical architecture design (system architecting). This direct interconnection conforms to what was explained regarding the importance of observing the dynamic behaviour of design alternatives in early design stages.

Generally, business experts perform the requirement analysis and provide the design knowledge for the designers. Detail designs are also domain and case dependent. Hence, this paper assumes that the design knowledge is available and downstream engineers perform the detailed design. Indeed, this paper focuses on system design from the SE perspective. Nonetheless, design is an iterative process of design generation, evaluation, and redesign (Pahl et al.

Fig. 1 Framework structure


Systems engineering tasks
System logical architecture design
Requirement analysis
Product
System validation
Trade study
Detail design
Detail design validation
Published by the same authors
Object Oriented (OO)
Architecting guidelines
Finite Machine State (FMS)
System State Meta Model - SSMM
Design knowledge
Trade study
Focus of this paper
Alternative generation algorithm
Feasibility checking algorithm
Qualitative alternatives
Optimization-Quantification
Linear programming
Automatic Layout Design Program
Quantitative alternatives
SSMM
Executable alternatives
Not satisfying the requirements OR can improve the KPIs
Simulation results
Satisfying the requirements OR stopping criteria is met
Validated alternatives

2007). Hence, the trade study can be repeated several times during the design process. Thus, this research also addresses the trade study in the framework. Therefore, from SE perspective, the required tasks in the proposing framework are given below and highlighted in Fig. 1.

1. System logical architecture design.
2. System validation (validation of design alternatives in system level).
3. Trade study.

From the design perspective, the model of the logical architecture should indicate the allocation of the design requirements to the subsystems. In the CES design context, the simulation-based assessments mainly aim to examine the dynamic behaviour of the design alternatives to observe/validate their conformance with the requirements, yet as mentioned, simulation-based approaches are mostly used very late when the important design decisions are already taken. Therefore, in this framework, the trade study takes into account the simulation (validation from an SE perspective) results for evaluating the design alternatives at the system level. This helps to select a better alternative for the detail design. The proposed framework is implemented in Matlab-Simulink. However, the framework can be implemented in other similar platforms as well.

3.2 Framework requirements

Table 1 demonstrates how the tasks of the proposed framework (set of How-Means) can satisfy the research objectives. This cross-examination helps to define the explicit requirements from the framework.

The system architecture is different from the design alternatives. In fact, the system architecture model can stand as a meta-model such that different alternatives conform to that. Yet, it is required to generate different alternatives that their specifications differentiate them, while a design alternative is feasible if it meets the requirements and satisfies the constraints. A meta-logical architecture opens avenues to generate the design alternatives algorithmically. Accordingly, the

Table 1 Correspondence of the framework tasks and the research objectives

Objectives	Framework tasks	How-means
1. Managing the complexity of the design knowledge by providing a prescriptive approach for structuring the design knowledge	1. System logical architecture design	(a) The framework should propose certain architecting guidelines to develop system logical architecture in terms of its decomposition to subsystems, allocating the design requirements to them, and designing subsystems interface
2. Enabling integrated design by having a holistic approach in terms of designing both non-physical (operational policies) and physical aspects of the system concurrently. The holistic approach should allow observation of the impact of the design decisions on the dynamic behaviour of the system	1. System logical architecture design 2. System validation	(b) The logical architecture should comprise of the dynamic essence of a CES. Hence, the architecting guidelines should allow addressing all types of design requirements in the logical architecture, both the operational policies and the physical aspects of the system (c) Validation should allow observing/capturing the dynamic behaviour of the design alternative by its simulation. Thus, the logical architecture artefact should be executable for the purpose of observing the impact of 'operational policy design' on system behaviour
3. Enabling integrated (multidisciplinary) design to facilitate achieving the design consistency easier	1. System logical architecture design 2. System validation 3. Trade study	(d) The logical architecture should provide modules for different discipline design in one model (e) Proposed mechanism for the trade study should apply the design constraints in the alternatives (f) The logical architecture artefact should be executable such that it allows observing the impact of different discipline designs on system behaviour/performance These features jointly allow identifying possible inconsistencies between different disciplines in early design stages

required artefacts in the framework along with their essential features are summarized as follow;

1. Architecting guidelines to develop the system's logical architecture.
2. Executable Meta-logical-architecture;

    (a) Includes design modules that allow integrated (multidisciplinary) design.
    (b) Embodies operational and physical design requirements that realize the holistic design approach.
    (c) Captures the dynamic behaviour of design alternatives.
    (d) Design alternatives are its instances.
    (e) Formulates MoEs for the purpose of design assessment/improvement

3. A mechanism that generates design alternatives conforming to the logical architecture.
4. A winnowing mechanism that identifies non-feasible alternatives for early identification of design inconsistencies.
5. A model that applies the design constraints on a feasible alternative.

3.3 Framework overview

Figure 1 demonstrates the overall structure of the proposed framework. One of the main artefacts of this framework is named System-State-Meta-Model (SSMM). Same authors recently published a modelling approach to develop SSMM, which is an executable Meta-logical-architecture and allows modelling a CES in a holistic manner (explained shortly) (Abdoli and Kara 2017a).

'Design knowledge database' stores the possible genres for each design requirement. The 'alternative generation' algorithm configures possible design alternatives by the allocation of design options (genres) to the design requirements. The 'feasibility checking' algorithm crosses off the infeasible alternatives, when the configuration of selected genres does not generate a feasible system alternative.

The framework utilizes certain optimization–quantification models, which seek to develop a numerical model for a feasible qualitative alternative, as shown in Fig. 2. The optimization–quantification models provide the initial but not validated values for Key Performance Indicators (KPIs) or MoEs. This framework uses linear programming (LP) and automated layout design program (ALDeP) as its optimization–quantification models, which the later one may only be used for those CESs that need a layout design.

Finally, the numerical model of an alternative is coupled with the SSMM such that the alternative can be simulated with the SSMM to observe its behaviour. The simulation can

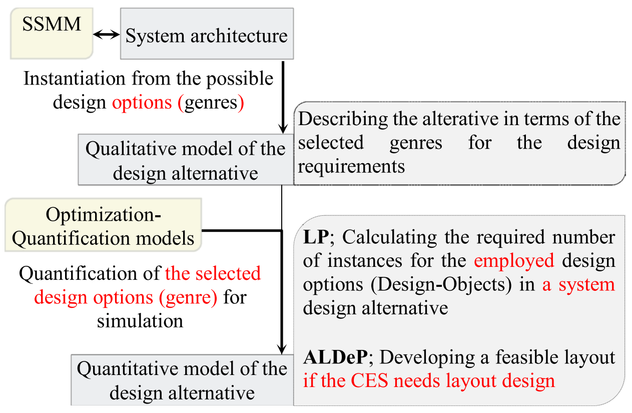

SSMM
System architecture
Instantiation from the possible design options (genres)
Describing the alterative in terms of the selected genres for the design requirements
Qualitative model of the design alternative
Optimization-Quantification models
Quantification of the selected design options (genre) for simulation
LP; Calculating the required number of instances for the employed design options (Design-Objects) in a system design alternative
ALDeP; Developing a feasible layout if the CES needs layout design
Quantitative model of the design alternative

Fig. 2 Evolution of a design alternative in the framework

validate the optimization–quantification results, which can be utilized in the trade study to evaluate/improve the alternatives and select a better one for detail design.

3.4 Framework artefacts and their interconnection

3.4.1 SSMM

The explained modelling approach in this section (published by the same authors) satisfies the first and second How-Means (A and B) from the proposed framework. The approach is briefly described in the current paper and interested readers are referred to the published work for more details.

OO modelling approach was used for managing the complexity of the design knowledge and accordingly achieving architecting guidelines, especially for having a holistic approach in modelling the system architecture, as given below.

1. Abstracting the CESs to abstract processes and acknowledging them as abstract classes that reflects the item state transformation in a CES (from input to output).
2. Acknowledging the sub-process process concept (factual process) as an abstract class.
3. Acknowledging the item concept as an abstract class.
4. Acknowledging process enablers as abstract classes.
5. Subclasses should be derived from the sub-process abstract class equivalent to the factual processes; sub-process.class
6. Subclasses should be derived from the related enabler abstract classes equivalent to the special types of enablers; enabler.class
7. Design requirements (DRs) from the enablers are addressed as design attributes of the related enabler.class.

8. Subclasses should be derived from item abstract class equivalent to the factual items; item.class
9. Formulating the system use case as a function of the required abstract processes and sub-processes for each factual item (process flow).
10. Addressing interaction attributes; 'Item Arrival' and 'Item Departure' attributes for sub-process abstract class and also 'Required processes' attribute for the item abstract class. The latter one is valued by the aforementioned use case formulation.

According to the guidelines, a CES logical architecture is modelled by developing its class diagram, fulfilling the first identified research gap. The first six guidelines make an abstraction hierarchy and decompose the CES to develop the structure of the logical architecture. The design attributes (DRs) are the translation of system requirements that are allocated to the enabler.classes based on the given architecting guidelines. Hence, the guidelines satisfy the realization of How-Means A in the framework. The explained abstraction and hierarchical decomposition allow addressing all types of DRs in the logical architecture; both operational and physical requirements (partial realisation of How-Means-B). In this approach, a process can have any type of physical or operational enabler, so the process concept covers a broad range and is not limited to the transforming material or energy and so on. In the early design stages, the available information is limited to knowing the genres of a design solution (Dauby and Dagli 2011). The design options (genres) for a DR are called design-objects.

The aforementioned guidelines mainly contribute to visualizing the holistic approach by the joint demonstration of the dynamic aspects of a CES and all types of its DRs in its logical architecture. Yet, a true realization of the holistic approach requires a proper modelling formalism such that can capture the dynamic aspects of the system by simulating its behaviour. The authors proposed establishing the logical-architecture as a Finite State Machine (FSM) model to achieve an executable Meta-logical-architecture, which was called the SSMM. The main idea relied on modelling a CES as a machine such that the DRs were considered equivalent to the states (design-requirement-state) of the machine with respect to its structure (according to the explained hierarchal decomposition). Each design-requirement-state encompasses certain sub-states representing its possible design-objects (object-states). Each object-state carries its own state-function for modelling its dynamic behaviour in its discipline. In return, the SSMM could allow the realization of integrated (multidisciplinary) design. These state-functions were called object-state-functions. Hence, the architecting guidelines resultant from OO modelling were mapped into FSM formalism, as given below:

1. Classes are mapped to states.
2. Associated classes to a higher-level class are mapped as sub-states of the corresponding supper state.
3. Item.class, process.class, and enabler.class are mapped to, respectively, item.state, process.state, and enabler.state.
4. Design attributes of an enabler.class are mapped to design-requirement-states of the equivalent enabler-state.
5. Design-objects of a DR are mapped to sub-states (object-state) of the corresponding design-requirement-state.
6. 'Item Arrival' and 'Item Departure' attributes are mapped to variables of the corresponding process.states.
7. 'Required-processes' attribute is mapped to a constant of the corresponding item. states.

Accordingly, a design alternative can be configured by activating one object-state for each and all of the design-requirement-states. Thus, SSMM is meta-model such that all alternatives conform to that. Alternatives can vary in their activated design-objects for their operational policies. As a result, the operational policies are not dictated to the SSMM (or being its invariant property). This realizes How-Means B, which serves to fulfil the third identified research gap.

The previous work of the authors mainly focused on explaining the core of the modelling approach. Yet, elaboration on addressing the specification of the design knowledge in the SSMM was left as the future research, for instance whether a design alternative meets the design constraints or how good it is in satisfying the KPIs compared to other alternatives. Hence, a design alternative should carry certain level of quantification such that allows alternative's simulation, for instance number of equipment. Therefore, it is needed to establish a systematic interconnection between design knowledge and SSMM to realize the simulation of the alternatives by SSMM. Hence, the new introduced concepts in the current paper can be summarized into; developing the optimization–quantification models, introducing supporting algorithms, explaining a full detail approach for leveraging the SSMM to an executable artefact, developing the interconnections between the SSMM, optimization–quantification models, and supporting algorithm as shown in Fig. 1. All the figures and tables (including the case study) in this paper are newly introduced in this current work of the authors.

3.4.2 Design database

Generally, decision making requires certain parameters to perform different analysing activities. Hence, it is crucial to define a proper set of parameters, including constants and variables, such that they provide the required basis for data interoperability between the framework artefacts

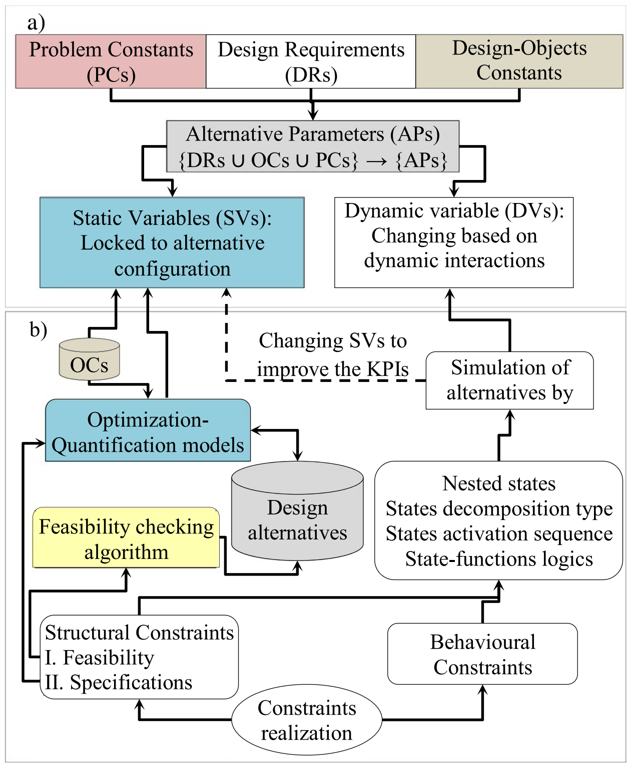

a)
Problem Constants (PCs)
Design Requirements (DRs)
Design-Objects Constants
Alternative Parameters (APs) {DRs ∪ OCs ∪ PCs} → {APs}
Static Variables (SVs): Locked to alternative configuration
Dynamic variable (DVs): Changing based on dynamic interactions
b)
OCs
Changing SVs to improve the KPIs
Simulation of alternatives by
Optimization-Quantification models
Design alternatives
Nested states
States decomposition type
States activation sequence
State-functions logics
Feasibility checking algorithm
Structural Constraints
I. Feasibility
II. Specifications
Behavioural Constraints
Constraints realization

Fig. 3 **a** Taxonomy of design alternatives parameters. **b** Constraints realization in frameworks artefacts

(algorithms, optimization–quantification, SSMM). Figure 3a introduces a taxonomy for the involving parameters in the design alternatives in this framework. DRs and design-objects are already explained. Problem constants (PCs) refer to those specifications that are fixed irrespective to the design decisions, such as required throughput or total budget.

The design knowledge database stores certain objects equivalent to the process.classes, which are called process-objects. The database also stores objects equivalent to factual items, which are called item-object. The known characteristics of a factual item/process are addressed as attributes of its equivalent process-object/item-object, such as input batch size/available time window. Item-objects require an attribute clarifying their process flows. For simplicity, the constants of factual processes and items are called PCs. The design-objects are also stored as objects in the database and their specifications are addressed as their attributes (object constants: OCs). For example, defining the type of material handling equipment is a DR in a storing process of a warehouse, while 'Forklift' and 'Turret truck' are two possible design-objects for that DR. Hence, attributes of 'Forklift' and 'Turret truck' are defined as; operation time, required space, and fixed cost. The database acts as a hub between the framework artefacts, which allows for the exchange of data and results between the framework artefacts.

3.4.3 Parameters of design alternatives

In a specific alternative, alternative parameters (APs) are dependent on PCs, DRs, and OCs of the employed design-objects in that alternative, see Fig. 3a. PCs and DRs are fixed for all the alternatives. APs include two types of variables; static variables (SVs) and dynamic variables (DVs).

Static variables When certain design-objects are allocated to DRs, the OCs of the employed design-objects become the constants of the design alternative. However, these constants can vary from one alternative to another. For instance, the operation time in the aforementioned storing process is dependent on the decision regarding the selection of 'Forklift' or 'Turret truck'. Generally, SVs are a function of the alternative configuration. In this paper, the alternative configuration refers to the combination of the employed design-objects that are allocated to the DRs. optimization–quantification models formulate these SVs as variables for each specific alternative, which their values are fixed after optimization–quantification application in one design iteration.

Dynamic variables DVs contribute to capturing the dynamic behaviour of an alternative. From the simulation perspective, the start and finish times of a process are time events changing dynamically. Therefore, processes and particularly their enablers require DVs to demonstrate the availability status of enablers for operation. Likewise, a DV also demonstrates when a certain item finishes its required operation in a process and is ready for the next process.

3.4.4 Design constraints

This paper divides design constraints into two main groups; structural and behavioural, see Fig. 3b. The structural constraints are further divided into two groups; feasibility and specifications. The feasibility constraints build boundaries such that an alternative configuration is feasible if it is located within those boundaries. These boundaries are high level or qualitative design constraints (recall warehouse example). The specification constraints are mainly dependent on each specific design case. Yet, they should be satisfied to meet the specific requirements, such as the required throughput. This framework addresses structural constraints in the optimization–quantification models.

Incorporation of the architecting guidelines into FSM formalism satisfies addressing the behavioural constraints into the SSMM. For instance, if certain processes can run simultaneously, their relative decomposition is defined as; 'Parallel ~ AND'. Interested readers can find more details regarding addressing the behavioural constraints in the previous work of the authors.

Figure 3a, b show how APs and constraints encounter each other in the framework artefacts.

3.4.5 Alternative generation algorithm

The 'alternative generation algorithm' generates a design alternative using an array of variables, which are equivalent to the DRs. Each DR is represented with a unique variable as shown in Fig. 4 and (1). The ID is the associated number to a DR; '$\mathrm{DR_{ID}}$'. This array is a reference model that reflects the addressed DRs in the logical architecture and as a result, the qualitative alternatives are its instances.

$$\mathrm{CES}_{\mathrm{Qualitative-Alternative}_i} = \left[\mathrm{DR}_1, \dots, \mathrm{DR}_{\mathrm{ID}}, \dots, \mathrm{DR}_z\right]. \tag{1}$$

The algorithm allocates the design-objects to the related DRs. Hence, each design-object carries an attribute clarifying whether it can be employed in each unique DR. This attribute is called applicability attribute, which can be valued as shown in (2).

$$\begin{aligned} &\mathrm{Applicability-design-object}_{\mathrm{U.Key}} = \left(\mathrm{Applicability}_1, \dots, \mathrm{Applicability}_j, \dots, \mathrm{Applicability}_z\right),\\ &\mathrm{Applicability}_j = \begin{cases} 1; & \text{if design} - \text{object can be used in the } \mathrm{DR}_j \\ 0; & \text{otherwise} \end{cases} \quad \forall j \in Z. \end{aligned} \tag{2}$$

Design-objects are differentiated by carrying Uniqueness Key (U.Key). In return, an alternative is demonstrated in a codified form showing U.Keys of the employed design-objects for each DR, as shown in (3). The algorithm is written as a query such that searches the database and generates possible alternatives and stores them in the database, which are called Alternative-Configuration in this paper.

$$\mathrm{CES}_{\mathrm{Qualitative-Alternative}_t} = \left[(\mathrm{Design.object}_i.\mathrm{U.Key})_{\mathrm{DR}_1}, \dots, (\mathrm{Design.object}_j.\mathrm{U.Key})_{\mathrm{DR}_z}\right]. \tag{3}$$

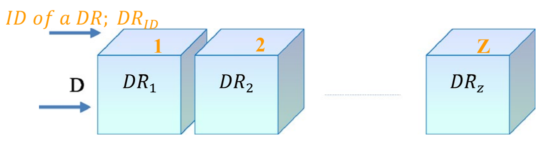


Fig. 4 Representing the DRs as an array of variables

3.4.6 Feasibility checking algorithm

Sometimes, a combination of individual design-objects may not necessarily result in a feasible system design alternative, recall the warehouse example. Hence, an algorithm is proposed such that examines an Alternative-Configuration with respect to the feasibility constraints and winnows the non-feasible alternatives. As a result, the remaining alternatives satisfy certain high-level qualitative (integration) constraints, this algorithm is called feasibility checking algorithm. The feasibility checking algorithm checks the feasibility constraints in the qualitative Alternative-Configurations before the application of optimization–quantification.

'Consistency attribute' is defined for each design-object that clarifies its consistency (compatibility) with other design-objects as shown in (4). K is the number of all stored design-objects in the database.

$$\begin{aligned} &\mathrm{Consistency-design-object}_{\mathrm{U.Key}} = \left(\mathrm{Consistency}_1, \dots \mathrm{Consistency}_i, \dots \mathrm{Consistency}_k\right)\\ &\mathrm{Consistency}_i = \begin{cases} 1; \text{ if the design} - \text{object is consistent with the design} - \text{object with U.Key of } i \\ 0; \text{ otherwise} \end{cases} \quad \forall i \in k \end{aligned} \tag{4}$$

In a specific Alternative-Configuration, the employed design-objects are a subset of all stored design-objects in the database. The feasibility checking algorithm refines the consistency attribute of a design-object with respect to the subset of employed design-objects in that specific Alternative-Configuration. The refined vector is called 'alternative-dependent consistency vector'. Thereupon, 'alternative-dependent consistency matrix' is built, which its rows are the 'alternative-dependent consistency vectors' as shown in (5). $\mathrm{C}_{\mathrm{o1o2}}$ represents the consistency of the employed design-object for DR_1 with the employed design-object for DR_2. The alternative feasibility is calculated as the production of matrix elements. A result of zero means that the Alternative-Configuration is not feasible, whereas one means feasible.

$$\text{Alternative dependent consistency matrix} = \begin{bmatrix} \text{NA} & C_{O1O2} & C_{O1O3} & \cdots & C_{O1Oz} \\ C_{O2O1} & \text{NA} & C_{O2O3} & \cdots & C_{O2Oz} \\ C_{O3O1} & C_{O3O2} & \text{NA} & \cdots & C_{O3Oz} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ C_{OzO1} & C_{OzO2} & C_{OzO3} & \vdots & \text{NA} \end{bmatrix}. \tag{5}$$

The feasibility checking algorithm is written as a follow-up of the alternative generation algorithm. Hence, the query first generates all possible Alternative-Configurations and then filters them to the feasible Alternative-Configurations. Therefore, the feasibility algorithm partially realizes the How-means E in the framework.

3.4.7 Quantification models

It is crucial to develop quantitative Alternative-Configuration to assure that it satisfies specific (numerical) constraints besides the structural ones. Recall the storing process example, in this case the number of 'Forklifts' or 'Turret truck' need to satisfy the required throughput. Although, optimization–quantification is applied on the qualitative Alternative-Configuration to obtain a quantitative model for it such that can be simulated with the SSMM, yet the optimization–quantification models also take the first steps to apply constraints on the quantified Alternative-Configuration.

Analytical formulation of the CESs can be fairly complicated (Zheng et al. 2016), so optimization has been mostly used for finding a good solution and not necessarily finding the global optimum (Roy et al. 2008). Hence, it is less critical to argue regarding the optimality of the formulated solution using a specific optimization method in this context. Therefore, in this framework, the term optimization–quantification is used instead of optimization. It is argued that the top three optimization approaches are: genetic algorithms, linear/quadratic programming, and simulated annealing. The application field of LP is so broad, including design optimization in engineering and resource allocation (Alfaris 2009). In CESs from a system level perspective, the variables are mainly related linearly in the objective function and constraints. Hence LP is selected for optimization–quantification in this framework. Yet a designer may use other satisfying optimization approaches.

Linear programming LP is a general technique that tries to model a problem with linear inequalities that are related to a linear objective function as shown in (6). The inequalities are problem constraints and the solution hold to them (if it exists).

$$\begin{aligned} &\text{Min} f \\ &f = \sum_{i=1}^{n} c_i \times x_i \\ &A \times x \le \text{b} \\ &\text{Aeq} \times x = \text{beq} \\ &\text{lb} < x < \text{ub}. \end{aligned} \tag{6}$$

The decision variables are $x_1, \ldots, x_n$. LP calculates (instantiates) the required number of the employed design-objects in an Alternative-Configuration. The quantified value of design-objects (x) is called 'Required number of instances. LP' and is coupled with the corresponding U.Key in the qualitative model of an Alternative-Configuration as shown in (7).

$$\begin{aligned} &\text{CES}_{\text{Qualitative-Alternative}_t} = \left[(\text{design.object}_\text{i}.\text{U.Key})_{\text{DR}_1}, \ldots, (\text{design.object}_\text{j}.\text{U.Key})_{\text{DR}_z} \right] \\ &\qquad\qquad\qquad\qquad \downarrow \text{LP} \\ &\text{CES}_{\text{Quantitative-Alternative}_t} = \left[\begin{array}{l} (\text{desig.object}_\text{i}.\text{U.Key},\ \text{required number of instances.LP})_{\text{DR}_1}, \\ \ldots, (\text{desig.object}_\text{j}.\text{U.Key},\ \text{required number of instances.LP})_{\text{DR}_z} \end{array} \right]. \end{aligned} \tag{7}$$

The objective function can be formulated to return the KPIs, such as cost minimization. The specific constraints can be modelled as LP constraints. PCs determine the 'b' and 'beq' in the LP formulation, such as a maximum budget. However, it is not very easy to find a globally optimal solution that maximizes multiple conflicting objectives. There are different approaches to deal with this issue, such as weighting the objectives or finding a set of solutions and the decision regarding prioritizing the objectives is made later. This research uses the first approach because it is simpler and more importantly the optimization results are validated later by simulation. In the CES design context, there are usually two typical competing objectives; cost and throughput. If one of them has a limit (e.g. throughput) that can be modelled as a constraint and the other (e.g. cost) as the primary objective in the objective function.

Certain constraints can be defined to satisfy the system level requirements. From the theory of constraints perspective, system goal achievement is constrained to its weakest sub-system, for instance a system throughput is limited to its bottleneck. Therefore, some constraints should be defined to realize the system throughput in each process.

Including all DRs (particularly operational policies) may result in NP formulation. Although there are approaches that allow formulating operational policies in an optimization model, such as queue theory, they are considerably complicated (Roy et al. 2008); this makes the simulation as the most applied approach for capturing the behaviour of CESs. Hence, only the quantifiable DRs are addressed in optimization–quantification models (e.g., required number of equipment), while the simulation of quantified Alternative-Configurations allows evaluating their KPI as a function of the dynamic interactions between the employed design-objects for all the DRs.

The LP model should not include any information about the design-objects. Thus, the A, Aeq, and C are needed to be formulated as variables, which are OCs of the employed design-objects in a specific Alternative-Configuration. In return, the LP model can be used for different alternatives.

Automated layout design program (ALDeP) Some CESs may require the layout design, such as warehouses, so it may be needed to address the layout design in their optimization–quantification step. Hence, the proposed framework can be used for CESs that require/do not require the layout design without losing the framework generality. Layout design approaches can be divided into two main groups; construction and improvements algorithms. The former group builds a block layout by iteratively adding departments. The latter group seeks to improve an initial block layout incrementally. This research uses ALDeP because; it is a constructive approach, generates multiple layouts, and also the final layout will not be an odd shape. Explaining the detail of ALDeP is out of the scope of this paper, hence interested readers are referred to the related literature (Hopp and Spearman 2011). ALDeP requires several inputs such as a number of departments, department areas, and departments' relationships. Certain parts of these inputs are known and belong to PCs, such as the number of departments. Yet, some inputs, such as department area, are a function of Alternative-Configuration and belong to SVs. ALDeP generates layout(s) for the quantified model of an alternative (quantified by LP). The ALDeP determines the department shapes for each process and their relative position. The developed layout is stored in the Alternative-Configuration.

Finally, application of the optimization–quantification models jointly with feasibility checking algorithm realize both structural and specific constraints on the alternatives (realization of How-Means E).

3.4.8 Simulation of the SSMM

The SSMM has nested states, demonstrating a hierarchical structure. The higher-level states are independent of the design solution and model the system structure and its design requirements. The lower-level states are solution dependent, which demonstrate the decisions for choices of the DRs. The explained decomposition hierarchy based on OO modelling builds nested stated in SSMM as shown in Fig. 5.

The previous work of authors suggested defining functions for each object-state to embody its design specifications in it. Yet, it was not elaborated on what data should be included in the state-functions, how they exchange data to realize the simulation of an alternative with SSMM and so on. From discrete event simulation (DES) perspective, the model components should be informed regarding the events to act accordingly to simulate the dynamic behaviour of the model. In DES, the variables play a key role in update propagation. An Alternative-Configuration is simulated by coupling its 'quantitative model' with the 'SSMM'. Thus, it is essential to have a proper interconnection between the APs and the SSMM. Therefore, three issues are joint-discussed in this section; first identification of critical time events, second defining certain variables that derive those events, and third interconnection of variables for update propagation during the simulation.

It is worth drawing attention to some points at the beginning of this section. First, system engineers have a system view and can visualize the interactions in the system level. Hence, system engineers can define how object-state-functions should interact with each other in terms of exchanging data. Second, this section explains those logics that object-state-functions include to realize the alternatives simulation in a system level. However, the detail level of addressed logics in the state-functions depends on the required granularity in the simulation.

As shown in Fig. 6, the object-state-functions contain two levels of information. The first level is those logics that embody the dynamic behaviour of the object-state in its discipline from a system level perspective. These logics contribute to early modelling of the dynamic behaviour of an object-state prior to moving to the detail design. These logics can be modelled with state modelling of the design-object as well, as shown in the bottom section of Fig. 5. DVs contribute to capturing the dynamic behaviour of an alternative hence they are embodied as inputs-outputs of the object-state-functions. Accordingly, DVs can play as interfaces between different object-state-functions. However, an object-state-function can also get certain inputs from the PCs, OCs or SVs as well. The explained concepts in this paper belong to the first information level. The second is those logics that address the detail design of a design-object.

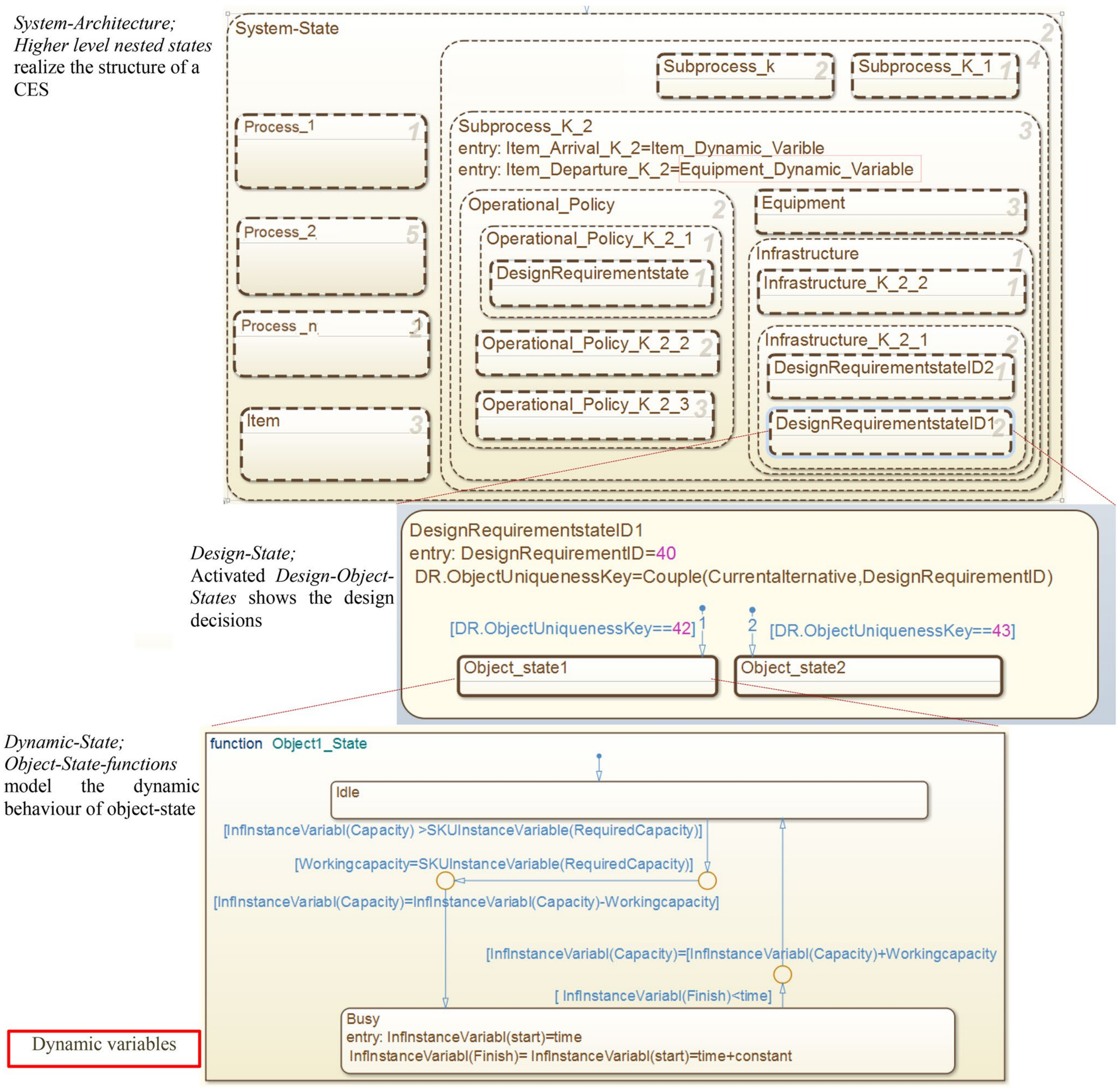

System-Architecture; Higher level nested states realize the structure of a CES
System-State
Process_1
Process_2
Process_n
Item
Subprocess_k
Subprocess_K_1
Subprocess_K_2
entry: Item_Arrival_K_2=Item_Dynamic_Varible
entry: Item_Departure_K_2=Equipment_Dynamic_Variable
Operational_Policy
Operational_Policy_K_2_1
DesignRequirementstate
Operational_Policy_K_2_2
Operational_Policy_K_2_3
Equipment
Infrastructure
Infrastructure_K_2_2
Infrastructure_K_2_1
DesignRequirementstateID2
DesignRequirementstateID1
Design-State; Activated Design-Object-States shows the design decisions
DesignRequirementstateID1
entry: DesignRequirementID=40
DR.ObjectUniquenessKey=Couple(Currentalternative,DesignRequirementID)
[DR.ObjectUniquenessKey==42]
[DR.ObjectUniquenessKey==43]
Object_state1
Object_state2
Dynamic-State; Object-State-functions model the dynamic behaviour of object-state
function Object1_State
Idle
[InfInstanceVariabl(Capacity) >SKUInstanceVariable(RequiredCapacity)]
[Workingcapacity=SKUInstanceVariable(RequiredCapacity)]
[InfInstanceVariabl(Capacity)=InfInstanceVariabl(Capacity)-Workingcapacity]
[InfInstanceVariabl(Capacity)=[InfInstanceVariabl(Capacity)+Workingcapacity
[InfInstanceVariabl(Finish)<time]
Busy
entry: InfInstanceVariabl(start)=time
InfInstanceVariabl(Finish)= InfInstanceVariabl(start)=time+constant
Dynamic variables

Fig. 5 SSMM realizes system architecture, design state, and dynamic state

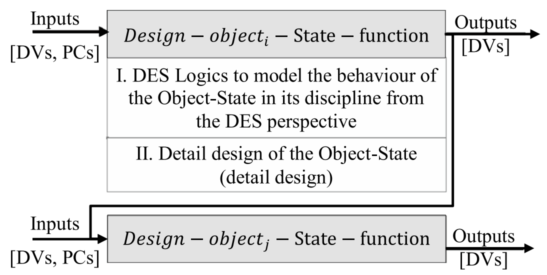

Inputs [DVs, PCs]
$Design - object_i - \text{State} - \text{function}$
Outputs [DVs]
I. DES Logics to model the behaviour of the Object-State in its discipline from the DES perspective
II. Detail design of the Object-State (detail design)
Inputs [DVs, PCs]
$Design - object_j - \text{State} - \text{function}$
Outputs [DVs]

Fig. 6 Embodied concepts in the objects-state-functions

The down-stream engineers can establish the detail design of an object-state in its object-state-function, yet they are aware that the encapsulated detail design should update the pre-defined ‘outputs’. Hence, the detail design can solidify the simulation results after certain design iterations. The object-state-functions can belong to any discipline, such as mathematical expressions or algorithmic procedures. Thus, the object-state-functions can stand as design modules for different disciplines; this realizes the essence of integrated (multi-disciplinary) design in the SSMM and satisfies the How-Means-D.

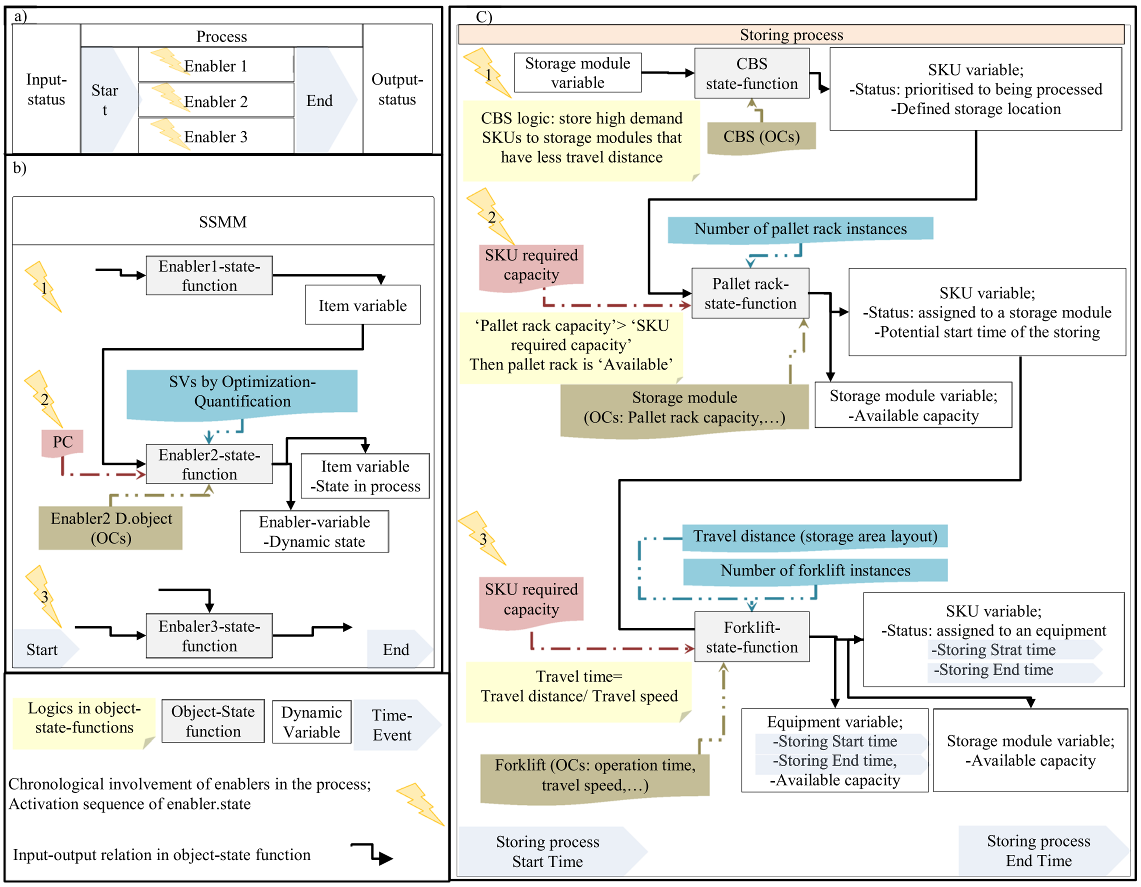
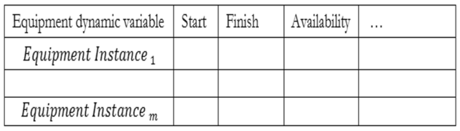

a)
Process
Input-status
Start
Enabler 1
Enabler 2
Enabler 3
End
Output-status
b)
SSMM
1
Enabler1-state-function
Item variable
2
SVs by Optimization-Quantification
PC
Enabler2-state-function
Item variable -State in process
Enabler2 D.object (OCs)
Enabler-variable -Dynamic state
3
Enbaler3-state-function
Start
End
Logics in object-state-functions
Object-State function
Dynamic Variable
Time-Event
Chronological involvement of enablers in the process; Activation sequence of enabler.state
Input-output relation in object-state function
C)
Storing process
1
Storage module variable
CBS state-function
SKU variable; -Status: prioritised to being processed -Defined storage location
CBS logic: store high demand SKUs to storage modules that have less travel distance
CBS (OCs)
2
Number of pallet rack instances
SKU required capacity
Pallet rack-state-function
SKU variable; -Status: assigned to a storage module -Potential start time of the storing
'Pallet rack capacity'> 'SKU required capacity' Then pallet rack is 'Available'
Storage module (OCs: Pallet rack capacity,…)
Storage module variable; -Available capacity
3
Travel distance (storage area layout)
Number of forklift instances
SKU required capacity
Forklift-state-function
SKU variable; -Status: assigned to an equipment -Storing Strat time -Storing End time
Travel time= Travel distance/ Travel speed
Equipment variable; -Storing Start time -Storing End time, -Available capacity
Storage module variable; -Available capacity
Forklift (OCs: operation time, travel speed,…)
Storing process Start Time
Storing process End Time

Fig. 7 **a** Process dynamic events. **b** interconnection of APs and framework artefacts from DES perspective. **c** an example of APs interconnection for simulation



Equipment dynamic variable	Start	Finish	Availability	…
Equipment Instance $_1$				
Equipment Instance $_m$				

Fig. 8 Example of enabler dynamic variable

Figure 7 shows the interconnection between the framework artefacts (SSMM, ALDeP, LP, databases) and APs from a DES perspective by demonstrating how the simulation happens using APs.

As shown in Fig. 7a, each process has two main events; Start and Finish. Availability of the process enablers is a key factor in the process operation. Hence, enabler-states require a variable clarifying their dynamic state in terms of their availability status. Recall the given model in (7). The enabler variable should clarify the dynamic state of each of the instances of an employed design-object. This research suggests defining the DVs as a structure with variable size as shown in Fig. 8.

SSMM should be able to simulate different Alternative-Configurations that employ different design-objects for one specific enabler; therefore, the enablers' DVs should be independent of the design-objects (e.g., Forklift and turret truck). Hence, the DVs are defined in the process.state or design-requirement-state and not at the object-state level as shown in Fig. 5.

A design-object can be employed for different DRs, see Fig. 9. Hence, the required number of instances of the same design-object can vary in different DRs. For instance, according to optimization–quantification, the required number of instances of 'Forklift' in stacking and storing processes are, respectively, 2 and 4. The required number of instances is an SV, which its interconnection with object-state-function is shown in Fig. 7b. For simplicity, the process of generating the required number of instances of a design-object is called instantiation. However, it is preferred that one object-state-function can be used for embodying the behaviour of similar object-states when they are employed in different design-requirements-states. Moreover, the object-state-function should be able to instantiate the design-object according to its employed DR (e.g. Forklift in stacking or

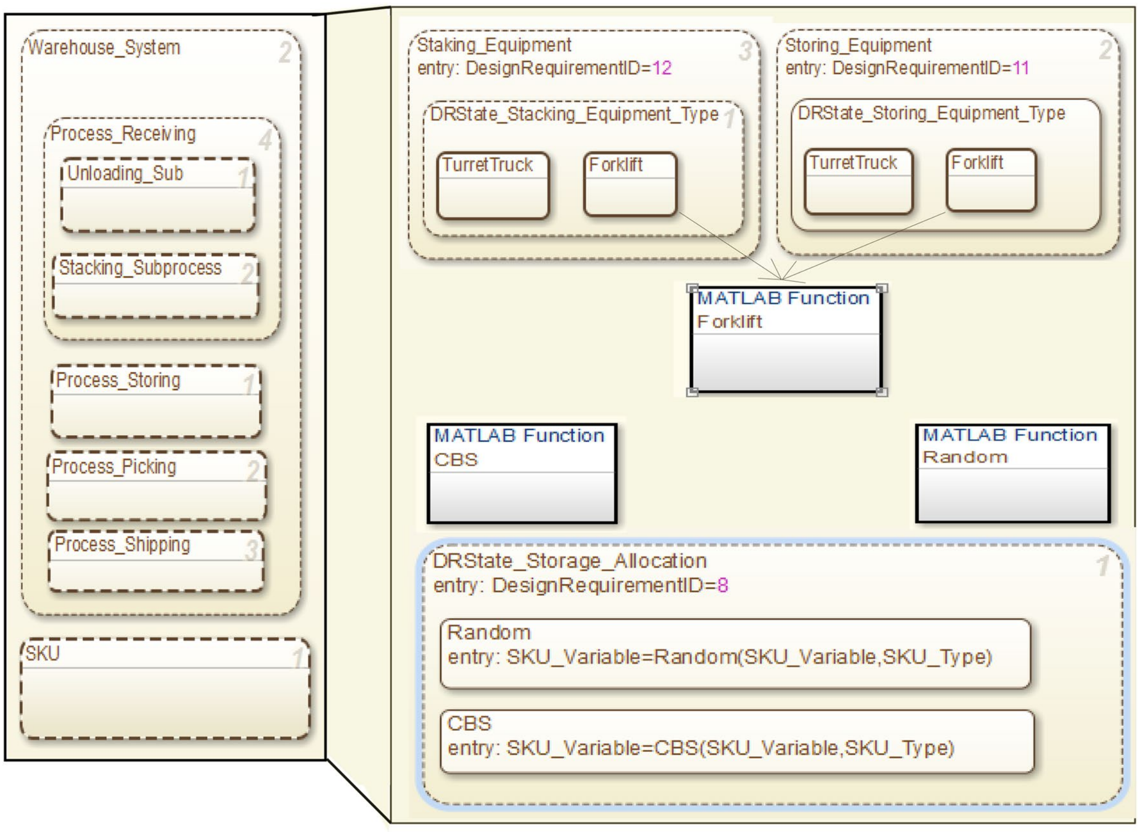

Warehouse_System
Process_Receiving
Unloading_Sub
Stacking_Subprocess
Process_Storing
Process_Picking
Process_Shipping
SKU
Staking_Equipment
entry: DesignRequirementID=12
DRState_Stacking_Equipment_Type
TurretTruck
Forklift
Storing_Equipment
entry: DesignRequirementID=11
DRState_Storing_Equipment_Type
TurretTruck
Forklift
MATLAB Function
Forklift
MATLAB Function
CBS
MATLAB Function
Random
DRState_Storage_Allocation
entry: DesignRequirementID=8
Random
entry: SKU_Variable=Random(SKU_Variable,SKU_Type)
CBS
entry: SKU_Variable=CBS(SKU_Variable,SKU_Type)

Fig. 9 Example of the design-requirement-states, their object-states and state-functions

```
- Read the 'Design Requirement ID' in the design-requirement-state and name it 'Current DR-state-ID'
- In the quantitate model of Alternative-Configuration find the DR that its ID ($DR_{ID}$) is equal to
  'Current DR-state-ID' and call that $DR_{ID}$ as $Current - DR_{ID}$
- Return the 'required number of instances' for the
  $Current - DR_{ID}$ and call it: $'Current -$ required number of instances$'_{ID}$
                                    Identifying the DR that employed the design-object

For i=1 to $'Current -$ required number of instances$'_{ID}$
    Enabler-Dynamic-Variable (i, 1) =i
End                                 Instantiation of the design-object based
                                    on its 'required number of instances'
```

Fig. 10 Structuring a dynamic variable with variable size

storing process). The procedure of instantiation is shown with a pseudo code in Fig. 10. Each design-requirement-state expresses the equivalent 'DR-ID' as an input to the object-state-function (DR-ID is shown as 'DesignReqiremntID' in Fig. 9). Hence, the object-state-function identifies the specific DR that employed the design-object. Then, the object-state-function instantiates the design-object according to the given value in 'Required number of Instances.LP'.

Likewise, if the CES required layout modelling, the object-state-functions retrieve the layout information from the quantitative model of an Alternative-Configuration. For instance, layout information can be used to calculate the travel time in material handling of storing process as shown in Fig. 7c.

From the system perspective, items hinge the processes together by leaving from one and entering into the next processes. Hence, the item concept also requires a variable clarifying the dynamic state of the items. Each real item is an instance of a specific item-object. Thus, the item variable also has a structure type with variable size and a similar instantiation procedure generates the same item-instances from one specific item-object.

If a process has a DR related to the operational policies, then an Alternative-Configuration should have an activated object-state for its design-requirement-state related to that operational policy. For instance, in the warehousing context, Class-Based-Storage (CBS) and Random storage are two operational policies for allocating Stock Keeping Units (SKUs) to storage modules. As shown in Fig. 9, the design-requirement-state related to the SKU allocation policy has two object-states; Random and CBS. Generally, each object-state-function takes the item-instances that require that specific process and manage them for different operational purposes. Each object-state-function may have its own operational logic. Hence, there are not dictated to the SSMM and their impacts on KPIs can be observed by simulation. As a result, SSMM allows having a holistic design in terms of making design decisions regarding the operational policy and physical aspects concurrently; this realizes the How-Means C.

The system engineer defines the dynamic interaction between enablers in terms of their chronological involvements in the process operation, see Fig. 7. Hence, the object-state-function of a posterior enabler.state gets certain required inputs from the prior enabler.states. As shown in Fig. 7c, the CBS-state-function defines the SKUs location by indicating that in SKU variable and communicates the results with the posterior enabler, which is pallet-rack as storage modules in this example.

Updating the dynamic state of enabler.state can be developed by simple functions following DES principles, as shown in the bottom section of Fig. 5. Indeed, the common updating principles can be written as general functions, which can be called in different object-state-functions.

The system architect defines those DVs that determine the start and finish time of the process. For example, the start and finish time of an equipment instance (e.g. Forklift) can be considered as start and finish time of the storing process for SKUs as shown in Fig. 7c. When the process finish time is elapsed, the item variable is updated showing that the item-instance has finished this process and is available for the next process. This procedure continues until the item-instance finishes its required process flow.

Certain variables can be defined in the SSMM to produce/formulate KPIs. Object-state-functions can contribute to dynamically update these KPIs and return their values

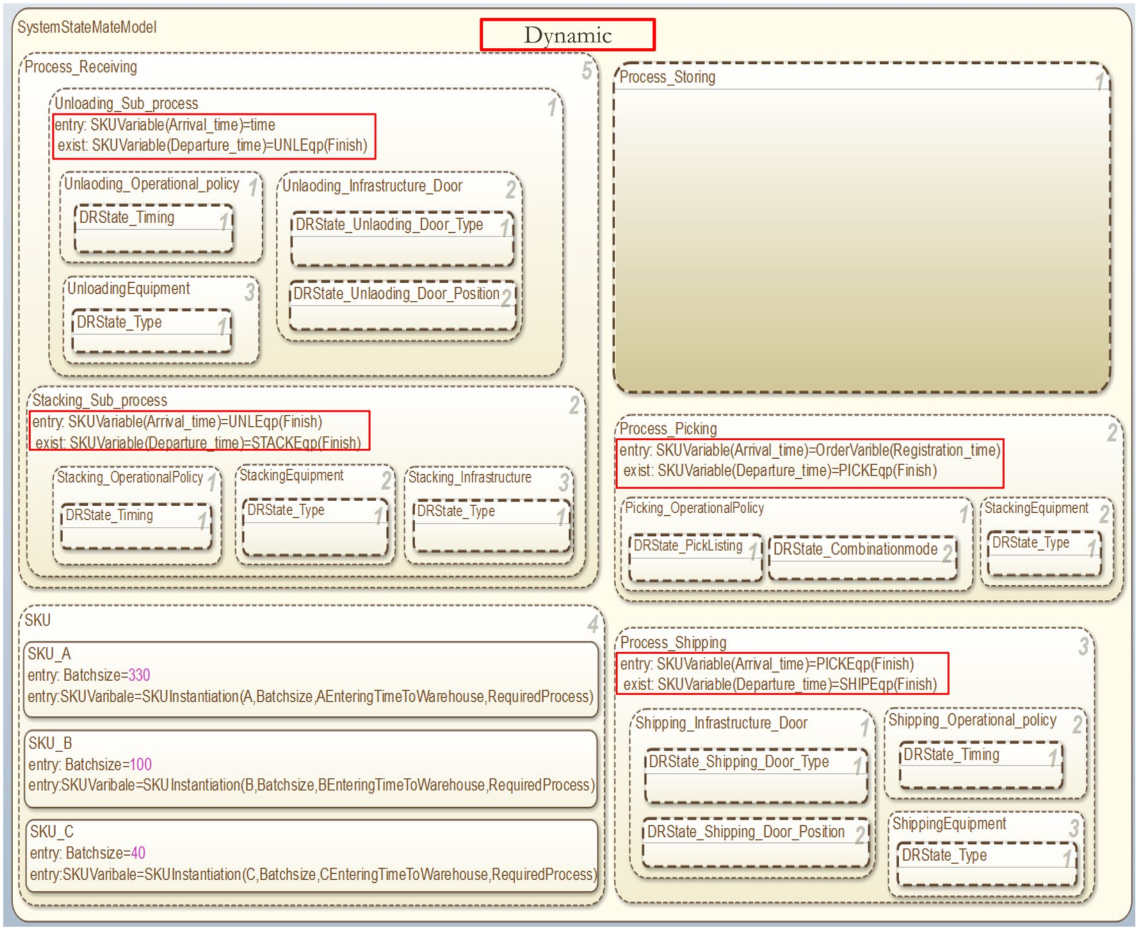

SystemStateMateModel
Dynamic
Process_Receiving 5
Unloading_Sub_process 1
entry: SKUVariable(Arrival_time)=time
exist: SKUVariable(Departure_time)=UNLEqp(Finish)
Unlaoding_Operational_policy 1
DRState_Timing 1
Unlaoding_Infrastructure_Door 2
DRState_Unlaoding_Door_Type 1
DRState_Unlaoding_Door_Position 2
UnloadingEquipment 3
DRState_Type 1
Stacking_Sub_process 2
entry: SKUVariable(Arrival_time)=UNLEqp(Finish)
exist: SKUVariable(Departure_time)=STACKEqp(Finish)
Stacking_OperationalPolicy 1
DRState_Timing 1
StackingEquipment 2
DRState_Type 1
Stacking_Infrastructure 3
DRState_Type 1
SKU 4
SKU_A
entry: Batchsize=330
entry:SKUVaribale=SKUInstantiation(A,Batchsize,AEnteringTimeToWarehouse,RequiredProcess)
SKU_B
entry: Batchsize=100
entry:SKUVaribale=SKUInstantiation(B,Batchsize,BEnteringTimeToWarehouse,RequiredProcess)
SKU_C
entry: Batchsize=40
entry:SKUVaribale=SKUInstantiation(C,Batchsize,CEnteringTimeToWarehouse,RequiredProcess)
Process_Storing 1
Process_Picking 2
entry: SKUVariable(Arrival_time)=OrderVarible(Registration_time)
exist: SKUVariable(Departure_time)=PICKEqp(Finish)
Picking_OperationalPolicy 1
DRState_PickListing 1
DRState_Combinationmode 2
StackingEquipment 2
DRState_Type 1
Process_Shipping 3
entry: SKUVariable(Arrival_time)=PICKEqp(Finish)
exist: SKUVariable(Departure_time)=SHIPEqp(Finish)
Shipping_Infrastructure_Door 1
DRState_Shipping_Door_Type 1
DRState_Shipping_Door_Position 2
Shipping_Operational_policy 2
DRState_Timing 1
ShippingEquipment 3
DRState_Type 1

Fig. 11 High-level snapshot of the SSMM developed for the warehouse

at the end of the simulation. For instance in a warehouse, shipping is the last process and 'Door' is the last enabler that is involved in the shipping process. Therefore, the object-state-function of the employed design-object for the 'Door' can return the total number of shipped orders during the simulation time.

The SSMM provided building blocks in different levels such as process-state, design–requirement-states, object-states, and object-state-functions. These building blocks can facilitate modelling those large-scale systems that have several similar subsystems such that the building blocks can be designed once and then be dragged and dropped in similar relevant levels multiple times.

Ultimately following the explained approach, the SSMM embodies three aspects of a CES; first: system architecture, second: decisions regarding the allocation of design options (genres) to the DRs for configuring a design alternative, and third: dynamic behaviour of the design alternative, see Fig. 5. The embodiment of such comprehensive concepts in the SSMM is the fruit of proper application of OO modelling, using proper modelling formalism (FSM), and their proper interconnection with optimization–quantification models and the supporting algorithms. Since all the defined How-Means are satisfied, hence the framework can satisfy the research objectives.

4 Case study

As proof of concept, the introduced framework is applied to a real industrial case. An international healthcare company looks to construct a new warehouse since the lease of the existing warehouse is about to end. The new warehouse receives three families of SKUs from two sources; namely a replenishment centre of a factory and the pharmacy department of the same company that is a separate business unit. The first SKU family is mainly ordered by hospitals with high daily demand (SKU-A). The second and third families are ordered by home patients in lower volumes (SKU-B and SKU-C). The SKUs come in different batch sizes in different weekdays. Likewise, the average order size for different SKU family is different. The details are shown in Appendices A1 and A2 in ESM.

4.1 Design meta-model and developing design alternatives

There is no need that the designers first make the class diagram of the CES and then map it to the FSM formalism; in fact, they can start making the SSMM following the explained guidelines. In this case study, the class diagram was developed on paper and model development was commenced in Matlab State flow library. A high-level snapshot of warehouse SSMM is illustrated in Fig. 11, according to the explained approach in Sect. 3.4.8.

In the warehouse, item-state of all SKU families change four times; from 'Inputs' to 'received SKU', then to 'stored SKU', then to 'picked ordered SKU' and finally to 'shipped ordered SKUs'. The receiving process includes two sub-processes; unloading and stacking; P_{11}, P_{12}. Storing, picking, and shipping are, respectively, shown as P_2, P_3 and P_4. The warehouse use case scenario is formulated as a function of SKUs' process flows as follows:

$$\begin{aligned}
W_{\mathrm{f}} &= \{f_{\mathrm{SKU}_1}, f_{\mathrm{SKU}_2}, f_{\mathrm{SKU}_3}\}\\
f_{\mathrm{SKU_j}} &= \{(\alpha_{j,i}, P_i)\};\ i \in \{1,\dots,4\},\ \ j \in \{1,2,3\},\\
\alpha_{j,i} &= \begin{cases} 1; & \text{if } P_i \text{ is needed for SKU}_j\\ 0; & \text{otherwise,}\end{cases}\\
f_{\mathrm{SKU}_2} &= \{(\alpha_{1,1}, P_1), (\alpha_{1,2}, P_2), (\alpha_{1,3}, P_3), (\alpha_{1,4}, P_4)\},\\
f_{\mathrm{SKU}_3} &= \{(\alpha_{1,1}, P_1), (\alpha_{1,2}, P_2), (\alpha_{1,3}, P_3), (\alpha_{1,4}, P_4)\},\\
\alpha_{1,i} &= \{(1)\} \quad \forall i \in (1,\dots 4) \to \alpha_1 = \{(1),(1),(1),(1)\},\\
\alpha_{2,i} &= \{(1)\} \quad \forall i \in (1,\dots 4) \to \alpha_2 = \{(1),(1),(1),(1)\},\\
\alpha_{3,i} &= \{(1)\} \quad \forall i \in (1,\dots,4) \to \alpha_3 = \{(1),(1),(1),(1)\},\\
\alpha_{4,i} &= \{(1)\} \quad \forall i \in (1,\dots 4) \to \alpha_4 = \{(1),(1),(1),(1)\},\\
P_i &= \{(\beta_{i,k}, \mathrm{SP}_k)\} \quad k \in (1,\dots K),\ \ \mathrm{SP_k}:\ \text{sub-process}_k\\
\beta_{\mathrm{i,k}} &= \begin{cases} 1; & \text{if sub-process } k \text{ is needed for } P_i\\ 0; & \text{otherwise}\end{cases}\\
P_1 &= \{(\beta_{1,1}, P_{11}), (\beta_{1,2}, P_{12})\}\\
\beta_{1,k} &= \{(1)\} \quad \forall i \in (1,2) \to \beta_1 = \{(1),(1)\}.
\end{aligned}$$

Since P_2, P_3 and P_4 do not have more than one sub-process, hence, $\beta_{i,k}$ was not shown for them. Therefore, the warehouse function was formulated as shown below.

$$w_{\mathrm{f}} = \left\{ \begin{array}{c} ((1,P_{11}),(1,P_{12}),(1,P_2),(1,P_3),(1,P_4))_1, ((1,P_{11}),(1,P_{12}),(1,P_2),(1,P_3),(1,P_4))_2,\\ ((1,P_{11}),(1,P_{12}),(1,P_2),(1,P_3),(1,P_4))_3 \end{array} \right\}.$$

The relation between the warehouse function and the reference model is that the DRs are defined for the enablers of the processes according to the explained decomposition hierarchy. The process enablers were defined as equipment, infrastructure, and operational policy. Using (1) the warehouse qualitative reference model was developed that represents the DRs and their ID-numbers as given below. Several design-objects are defined and stored in the database, as given in Appendix A3 in ESM.

$$\begin{aligned}\text{Warehouse}_{\text{Qualitative-Model}}:\ &[\text{Unloading}-\text{Equipment}-\text{type}_1,\ \text{Unloading}-\text{Infrastructure}-\text{door}-\text{position}_2,\\ &\text{Unloading}-\text{Infrastructure}-\text{door}-\text{type}_3,\ \text{Unloading}-\text{SKU}-\text{management}-\text{policy}-\text{timing}_4,\\ &\text{Staking}-\text{SKU}-\text{management}-\text{policy}-\text{timing}_5,\ \text{Stacking}-\text{Infrastructure}-\text{type}_6,\\ &\text{Stacking}-\text{equipment}-\text{type}_7,\ \text{Storing}-\text{SKU}-\text{management}-\text{policy}-\text{storage}-\text{allocation}_8,\\ &\text{Storing}-\text{SKU}-\text{management}-\text{policy}-\text{timing}_9,\ \text{Storing}\ -\text{Equipment}-\text{management}-\text{policy}-\text{combination}-\text{mode}_{10},\\ &\text{Storing}-\text{Equipment}-\text{type}_{11},\ \text{Storing}-\text{Infrastructure}-\text{storage}-\text{module}-\text{type}_{12},\\ &\text{Storing}-\text{Infrastructure}-\text{aisles}-\text{configuration}_{13},\ \text{Picking}-\text{Order management}-\text{policy}-\text{pick listing}_{14},\\ &\text{Picking}-\text{Equipment}-\text{management}-\text{policy}-\text{combination}-\text{mode}_{15},\ \text{Picking}-\text{Equipment}-\text{type}_{16},\\ &\text{Shipping}-\text{Infrastructure}-\text{door}-\text{type}_{17},\ \text{Shipping Infrastructure}-\text{door}-\text{position}_{18},\\ &\text{Shipping}-\text{Order management}-\text{policy}-\text{timing}_{19},\ \text{Shipping}-\text{Equipment}-\text{type}_{20}].\end{aligned}$$

4.2 Application of alternative generation and feasibility checking algorithms

A query (Matlab Script) is written to automatically perform the alternative generation and feasibility checking algorithms by taking into account the applicability and consistency attributes of the defined design-objects. The feasibility checking algorithm crossed more than 50% of the configured alternatives due to the inconsistency of their employed design-objects (10,500 generated alternatives were reduced to 4800 feasible alternatives). In fact, the risk of design failures was more than 50% if the decisions regarding employing the design-objects for different DRs were performed individually in different disciplines. This shows one of the benefits of the framework in terms of recognition of those alternatives that may lead to failures in later design stages. Due to space limitation, only two alternatives are shown in a coded form as examples.

$$\begin{aligned}&\text{Warehouse}_{\text{Qualitative-Alternative}_A}\\ &\quad=[1_1,3_2,5_3,7_4,7_5,10_6,11_7,13_8,7_9,16_{10},11_{11},\\ &\qquad 18_{12},20_{13},22_{14},16_{15},11_{16},6_{17},3_{18},8_{19},2_{20}]\\ &\text{Warehouse}_{\text{Qualitative-Alternative}_B}\\ &\quad=[1_1,3_2,5_3,7_4,7_5,10_6,11_7,14_8,7_9,15_{10},11_{11},\\ &\qquad 18_{12},20_{13},22_{14},15_{15},11_{16},6_{17},3_{18},8_{19},2_{20}].\end{aligned}$$

Recall CBS and random operational policies, likewise double command and single command are two possible operational options for storing/picking of the SKUs in and from the storage modules. In this warehouse, a maximum of five levels-heights for the storage modules is allowed and storing and picking from fifth level of storage aisles required more time. Due to high space cost, the project managers decided to have the five levels heights of the storage modules (similar to the existing warehouse). Two contributions of this framework are; addressing the operational policies in the SSMM and second; simulating the SSMM to observe the impact of the design decisions on the dynamic behaviour of the system. Hence, this section continues with the two above Alternative-Configurations to demonstrate the usefulness of these contributions.

4.3 Application of optimization–quantification on the design alternatives

4.3.1 LP application

The project managers preferred to minimize the cost while meeting the required throughput in terms of fulfilling customer orders. Thus, the LP is formulated such that the objective function returns the total cost. The cost includes; fixed cost and operational cost of the design-objects, and space cost.

Picking and shipping processes are triggered by customer orders. The company has an agreement with its customers to accept the orders in 5 h time window. Hence, the customer order specifications (See Appendix A1 in ESM) determine the bounds for LP constraints related to these two processes. However, unloading, stacking, and storing processes do not deal with customer orders. In fact, these three processes should keep up the pace with supply input rate, which is used to make bounds for LP constraints related to these three processes. The warehouse will be constructed close to a residential area and will only have 3 h in the early morning to receive the suppliers' inputs. The time horizon for each process is given in Appendix A2 in ESM. 130 working days is considered to include the operational cost in the LP formulation. The formulated LP model is shown below;

Table 2 ALDeP results

Process	Net required space (m^2)	Department shape ($m \times m$)
P_{11}	32	8×4
P_{12}	252	28×9
P_2	320	32×10
P_3	20	5×4
P_4	60	4×115

$$x_i = \begin{bmatrix} x_1 \\ . \\ x_{\mathrm{NQ}_I} \\ . \\ x_{\mathrm{NQ}_1} \\ . \\ x_{\mathrm{NQ}} \end{bmatrix}; \quad i = 1 : \mathrm{NQ},$$

Time interval (number of working days) = 130

Order time window = OTW

Supply time window = STW

SC = Space Cost

SKU_j = SKU type j; $j = 1 : 3$

$\mathrm{SKU}_{j,\mathrm{STW}}$ = Time window for accepting SKU_j from suppliers

$\mathrm{SKU}_{j,\text{Input batch size}}$ = Input batch size of SKU_j

$\mathrm{SKU}_{j,\text{Input rate}}$ = Average time interval to get a supply of SKU_j

$\mathrm{SKU}_{j,\mathrm{OTW}}$ = Time window for accepting orders of SKU_j from customers

$\mathrm{SKU}_{j,\text{Order size}}$ = Average order size of SKU_j

$\mathrm{SKU}_{j,\text{Order rate}}$ = Average time interval to get an order of SKU_j

PTH = process time horizon

Total number of quantifiable DRs = NQ

Total Number of Quantifiable DRs in unloading and stacking and storing = NQ_1

Total Number of Quantifiable DRs in picking and shipping = NQ_2

$\mathrm{NQ}_1 + \mathrm{NQ}_2 = \mathrm{NQ}$

x_i = Variable i in the LP

Dr_{x_i} = The 'DR – ID' for the x_i in the qualitative model

$\mathrm{Dr}_{x_i^{\mathrm{FC}}}$ = Fix cost(FC) of the employed design – object for the Dr_{x_i}

$\mathrm{Dr}_{x_i^{\mathrm{RS}}}$ = Required space(RS) for the employed design – object for the Dr_{x_i}

$\mathrm{Dr}_{x_i^{\mathrm{OC}}}$ = Operational cost(OC) of the employed design – object for the Dr_{x_i}

$\mathrm{Dr}_{x_i^{\mathrm{OT}}}$ = Operational time(OT) for the employed design – object for the Dr_{x_i}

$\mathrm{Dr}_{x_i^{\mathrm{capacity}}}$ = Working capacity of the employed design – object for the Dr_{x_i},

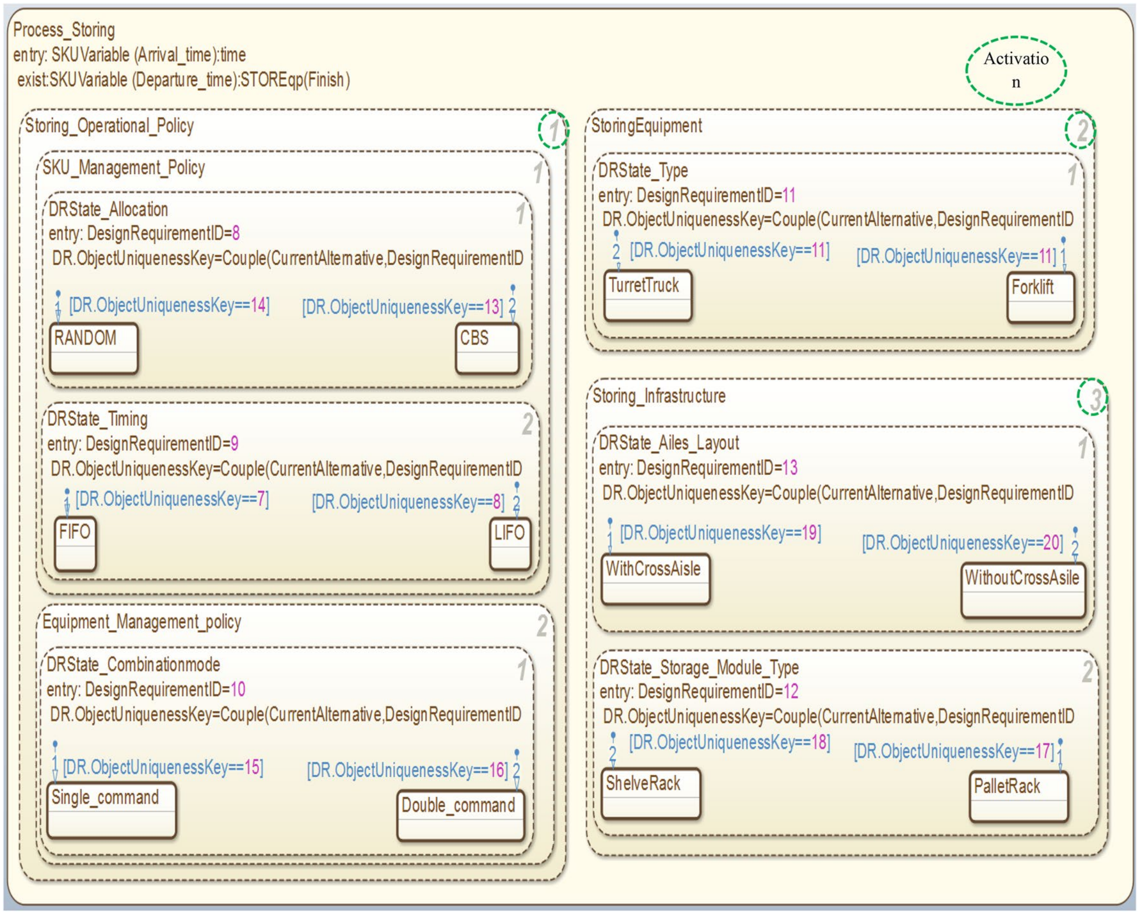

Process_Storing
entry: SKUVariable (Arrival_time):time
exist:SKUVariable (Departure_time):STOREqp(Finish)
Activation
Storing_Operational_Policy
SKU_Management_Policy
DRState_Allocation
entry: DesignRequirementID=8
DR.ObjectUniquenessKey=Couple(CurrentAlternative,DesignRequirementID
[DR.ObjectUniquenessKey==14]
[DR.ObjectUniquenessKey==13]
RANDOM
CBS
DRState_Timing
entry: DesignRequirementID=9
DR.ObjectUniquenessKey=Couple(CurrentAlternative,DesignRequirementID
[DR.ObjectUniquenessKey==7]
[DR.ObjectUniquenessKey==8]
FIFO
LIFO
Equipment_Management_policy
DRState_Combinationmode
entry: DesignRequirementID=10
DR.ObjectUniquenessKey=Couple(CurrentAlternative,DesignRequirementID
[DR.ObjectUniquenessKey==15]
[DR.ObjectUniquenessKey==16]
Single_command
Double_command
StoringEquipment
DRState_Type
entry: DesignRequirementID=11
DR.ObjectUniquenessKey=Couple(CurrentAlternative,DesignRequirementID
[DR.ObjectUniquenessKey==11]
[DR.ObjectUniquenessKey==11]
TurretTruck
Forklift
Storing_Infrastructure
DRState_Ailes_Layout
entry: DesignRequirementID=13
DR.ObjectUniquenessKey=Couple(CurrentAlternative,DesignRequirementID
[DR.ObjectUniquenessKey==19]
[DR.ObjectUniquenessKey==20]
WithCrossAisle
WithoutCrossAsile
DRState_Storage_Module_Type
entry: DesignRequirementID=12
DR.ObjectUniquenessKey=Couple(CurrentAlternative,DesignRequirementID
[DR.ObjectUniquenessKey==18]
[DR.ObjectUniquenessKey==17]
ShelveRack
PalletRack

Fig. 12 Storing-process-state

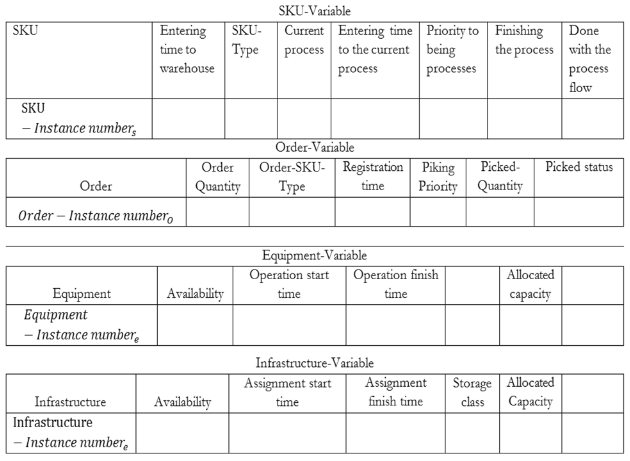
$$
\mathrm{c}_i^T = \begin{bmatrix} \left(\mathrm{Dr}_{\mathrm{V}_1^{\mathrm{FC}}} + \left(\mathrm{Dr}_{\mathrm{V}_1^{\mathrm{RS}}} \times \mathrm{SC}\right) + \left(\mathrm{Dr}_{\mathrm{V}_1^{\mathrm{OC}}} \times \mathrm{TIM} \times \mathrm{PTH}\right)\right) \\ \left(\mathrm{Dr}_{\mathrm{V}_i^{\mathrm{FC}}} + \left(\mathrm{Dr}_{\mathrm{V}_i^{\mathrm{RS}}} \times \mathrm{SC}\right) + \left(\mathrm{Dr}_{\mathrm{V}_i^{\mathrm{OC}}} \times \mathrm{TIM} \times \mathrm{PTH}\right)\right) \\ \left(\mathrm{Dr}_{\mathrm{V}_{\mathrm{NQ}_1}^{\mathrm{FC}}} + \left(\mathrm{Dr}_{\mathrm{V}_{\mathrm{NQ}_1}^{\mathrm{RS}}} \times \mathrm{SC}\right) + \left(\mathrm{Dr}_{\mathrm{V}_{\mathrm{NQ}_1}^{\mathrm{OC}}} \times \mathrm{TIM} \times \mathrm{PTH}\right)\right) \\ \left(\mathrm{Dr}_{\mathrm{V}_{\mathrm{NQ}}^{\mathrm{FC}}} + \left(\mathrm{Dr}_{\mathrm{V}_{\mathrm{NQ}}^{\mathrm{RS}}} \times \mathrm{SC}\right) + \left(\mathrm{Dr}_{\mathrm{V}_{\mathrm{NQ}}^{\mathrm{OC}}} \times \mathrm{TIM} \times \mathrm{PTH}\right)\right) \end{bmatrix},
$$

$$
\text{lower bound} = \mathrm{lb},\ \ \text{upper bound} = \mathrm{ub},
$$

$$
\mathrm{lb} < x < \mathrm{ub},\ \ \mathrm{ub} = [\mathrm{Infinite}]_{\mathrm{NQ}\times 1},\ \ \mathrm{lb} = \left[\mathrm{lb}_i\right]_{\mathrm{NQ}\times 1},
$$

$$
\mathrm{lb}_i = \frac{\left(\sum_{j=1}^{3}\left(\left(\frac{\mathrm{SKU}_{j,\mathrm{Input\ rate}} \times \mathrm{SKU}_{j,\mathrm{STW}}}{\mathrm{SKU}_{j,\mathrm{Input\ rate}}}\right)\right) \times \mathrm{Dr}_{x_i^{\mathrm{OT}}}\right)}{\left(\mathrm{Dr}_{x_i^{\mathrm{capacity}}} \times \mathrm{PTH}\right)} \quad \forall i \in 1 : \mathrm{NQ}_1,
$$

$$
\mathrm{lb}_i = \frac{\left(\sum_{j=1}^{3}\left(\left(\frac{\mathrm{SKU}_{j,\mathrm{Order\ size}} \times \mathrm{SKU}_{j,\mathrm{OTW}}}{\mathrm{SKU}_{j,\mathrm{order\ rate}}}\right)\right) \times \mathrm{Dr}_{x_i^{\mathrm{OT}}}\right)}{\left(\mathrm{Dr}_{x_i^{\mathrm{capacity}}} \times \mathrm{PTH}\right)} \quad \forall i \in \mathrm{NQ}_1 + 1 : \mathrm{NQ},
$$


SKU-Variable

SKU	Entering time to warehouse	SKU-Type	Current process	Entering time to the current process	Priority to being processes	Finishing the process	Done with the process flow
SKU – *Instance number*$_s$							

Order-Variable

Order	Order Quantity	Order-SKU-Type	Registration time	Piking Priority	Picked-Quantity	Picked status
Order – Instance number$_o$						

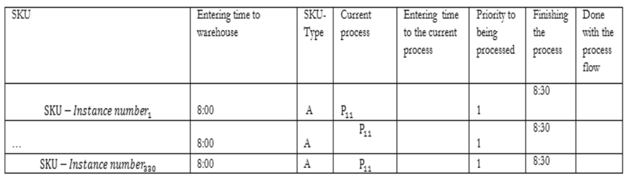
Equipment-Variable

Equipment	Availability	Operation start time	Operation finish time		Allocated capacity	
Equipment – Instance number$_e$						

Infrastructure-Variable

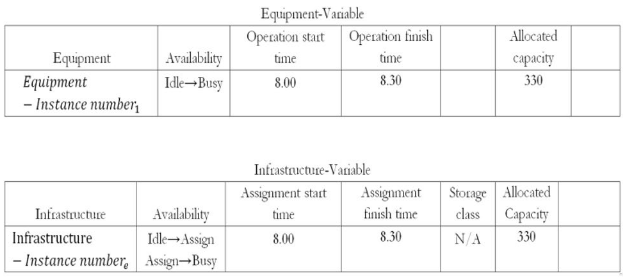
Infrastructure	Availability	Assignment start time	Assignment finish time	Storage class	Allocated Capacity	
Infrastructure – *Instance number*$_e$						

Fig. 13 Defined DVs for SKU, order, infrastructure and equipment



SKU	Entering time to warehouse	SKU-Type	Current process	Entering time to the current process	Priority to being processed	Finishing the process	Done with the process flow
SKU – *Instance number*$_1$	8:00	A	P_{11}		1	8:30	
…	8:00	A	P_{11}		1	8:30	
SKU – *Instance number*$_{330}$	8:00	A	P_{11}		1	8:30	

Fig. 14 Updating SKU-dynamic variable



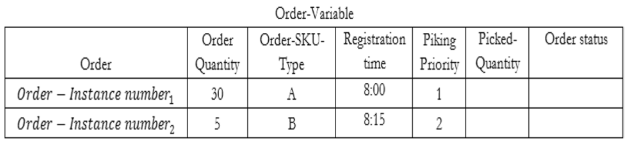
Equipment-Variable

Equipment	Availability	Operation start time	Operation finish time		Allocated capacity	
Equipment – Instance number$_1$	Idle→Busy	8.00	8.30		330	

Infrastructure-Variable

Infrastructure	Availability	Assignment start time	Assignment finish time	Storage class	Allocated Capacity	
Infrastructure – *Instance number*$_e$	Idle→Assign Assign→Busy	8.00	8.30	N/A	330	

Fig. 15 Updating equipment and infrastructure dynamic variables



Order-Variable

Order	Order Quantity	Order-SKU-Type	Registration time	Piking Priority	Picked-Quantity	Order status
Order – Instance number$_1$	30	A	8:00	1		
Order – Instance number$_2$	5	B	8:15	2		

Fig. 16 Updating order dynamic variable

$$f = \sum_{i=1}^{\mathrm{NQ}} c_i \times x_i, \quad \text{objective}: \quad \text{minimize } f.$$

The LP model retrieves each feasible qualitative Alternative-Configuration from the database automatically. Accordingly, the coefficients are automatically updated with respect to the employed design-objects in each Alternative-Configuration and LP results are written in it automatically. As mentioned, the optimization–quantification models quantify the quantifiable design-objects, so the design-objects related to the operational policies are not addressed in the LP formulation in this case. Hence, the LP results of the two given alternatives are similar because they only differ in their operational policies. In fact, the analytical formulation of the order picking belongs to NP-hard problem class when the operational policy is addressed in the formulation (de Koster et al. 2007). In this design case, the operator can only entre to the aisles from the right side and exit from the left side due to safety reasons. Moreover, the operators cannot have backward movement in the aisle. If the double command mode is selected for the order picking and storing processes, then the operator can pick an order from the pick list, if the allocated SKUs to the pick list is located in the same aisle that operator stored SKUs recently. Moreover, the SKUs should be on the left side of the operator in the aisle, so the operator does not need a backward movement. Many studies tried to model such policies with different approaches, such as queue theory. Yet, such complicated formulations include so many simplifications. Moreover, designing the entire warehouse requires modelling of operational policies for different processes, which can make the optimization formulation even more complicated, while there is no guarantee for finding the global optimal solution that also captures the dynamic interactions in the system as good as the simulation does. However, the proposed framework allows observing the impact of different operational policies in the dynamic behaviour of the system by means of simulation.

$$\begin{aligned}
\mathrm{Warehouse}_{\mathrm{Quantitative-Alternative}_A} = {} & [(1_1, 1), (3_2, \mathrm{NA}), (5_3, 1), (7_4, \mathrm{NA}), (7_5, \mathrm{NA}), (10_6, 90), (11_7, 3), (13_8, \mathrm{NA}), \\
& (7_9, \mathrm{NA}), (16_{10}, \mathrm{NA}), (11_{11}, 4), (18_{12}, 258), (20_{13}, \mathrm{NA}), (22_{14}, \mathrm{NA}), \\
& (16_{15}, \mathrm{NA}), (11_{16}, 4), (6_{17}, 1), (3_{18}, \mathrm{NA}), (8_{19}, \mathrm{NA}), (2_{20}, 3)] \\
\mathrm{Warehouse}_{\mathrm{Quantitative-Alternative}_B} = {} & [(1_1, 1), (3_2, \mathrm{NA}), (5_3, 1), (7_4, \mathrm{NA}), (7_5, \mathrm{NA}), (10_6, 90), \\
& (11_7, 3), (14_8, \mathrm{NA}), (7_9, \mathrm{NA}), (15_{10}, \mathrm{NA}), (11_{11}, 4), (18_{12}, 258), (20_{13}, \mathrm{NA}), \\
& (22_{14}, \mathrm{NA}), (15_{15}, \mathrm{NA}), (11_{16}, 4), (6_{17}, 1), (3_{18}, \mathrm{NA}), (8_{19}, \mathrm{NA}), (2_{20}, 3)].
\end{aligned}$$

4.3.2 ALDeP application

The ALDeP model was applied to generated alternatives to produce feasible layouts. The initial safety requirements are also applied in ALDeP such as allocating certain space between departments. Table 2 gives the calculated space for each process based on the employed design-objects in the two given alternatives. The ALDeP results are written in the quantitative model of the alternatives.

4.4 Embodiment of the design specifications to simulate a design alternative

Figure 12 shows the storing process state that embodies its required enablers and their DRs with a hierarchal structure. Such structure applied for other processes according to their defined enablers, DRs and design-objects. Although the available time windows for the processes were different in this warehouse, they technically could work together. Hence, the decomposition of the SSMM in the highest level was defined 'Parallel ~AND' (dashed lines). In return, the process.states can be active simultaneously. The process enablers can work simultaneously in each process. Hence, the decomposition of the process-states with respect to their enabler.states is defined as 'Parallel ~AND'. In all five processes, the operational policy enablers determine the way that the process should operate. Hence, those design-requirement-states that are related to the operational policy enablers have the first activation sequences, as shown in green circles in Fig. 12. The relative decomposition of the design-requirement-states with respect to their object-States are defined 'Exclusive ~ OR' (solid lines) because each design-requirement can only employ one design-object in a specific Alternative-Configuration. Certain DVs are defined for the SKUs, orders and enablers, which their structures are shown in Figs. 13, 14, 15 and 16.

In warehousing domain, the process flow starts with receiving the inputs from the suppliers and continues by converting them into SKUs. They move forward to being stored. The picking process is triggered by customer orders and the process flow finishes by shipping the orders. Hence, the simulated process flow in this case study is explained in two sections; 'supplier inputs into stored SKUs' and 'stored SKUs into shipped orders'.

4.4.1 Supplier SKUs into stored SKUs

It was described in the previous work of the authors how to address the CES interactions with its environment. Three main attributes of an SKU are addressed in each supplier sub-state as its constants (PCs); the SKU type, its supply rate, and the input batch size, see Fig. 11. Likewise, three main attributes of an order are addressed in each customer sub-state; SKU type, order rate, and average order size. Supplier-A sends SKU-A in input batches of 330. Accordingly, SKU-instantiation assigns a unique instance number to each of 330 instances of SKU-A. Likewise, 'SKU-type' and 'Entering time to warehouse' are written in the 'SKU-variable' as shown in Fig. 13. In this case study, all state-functions use the Matlab-Simulink clock as the time input.

All input SKU-batches require the unloading process. The activation sequence of unloading enablers is; (1) operational policy, (2) door (as infrastructure), (3) equipment. Alternative$_A$ employed 'FIFO-object' in the unloading process configuration. Therefore, the 'FIFO-state-function' gets the 'SKU-variable' as its input to sort available SKU-instances of the coming input batch for the unloading process. The sorting result is written in 'Priority to being processed' as shown in Fig. 14.

Afterwards, 'Door-type-state' is activated. 'Door-type-state' is a design-requirement-state with two object-states; 'Sunken-object' and 'Level-object'. The employed design-object for 'Door-type' is 'Sunken-object'. The 'Sunken-state-function' has several inputs including 'SKU-variable', 'Door-variable', and time. The 'Door-variable' follows the structure of 'Infrastructure-variable' that is given in Fig. 13. These variables are independent of the design-objects. In fact, 'Door-variable' serves both 'Sunken-object' and 'Level-object', because, 'Level-state-function' and 'Sunken-state-function' uses similar input/output variables.

The quantitative model of Alternative$_A$ shows that it requires one 'Sunken' door to fulfil the required throughput. Therefore, one instance of 'Sunken-object' is generated according to the explained instantiation procedure, see Fig. 10. The 'Sunken-state-function' gets 'SKU-variable' carrying 'Priority to being processed'. Accordingly, the 'Sunken-state-function' determines which SKU-instances can possibly enter into the warehouse. Yet, the unloading process cannot start until an instance of the unloading equipment is available. The 'Sunken-state-function' only changes the state of a Door-instance from 'available' to 'assign', because door availability is a priory condition for the unloading process.

The unloading equipment as a DR has two options 'Cargomatic' and 'Carne'. Alternative$_A$ is formulated based on using 'Cargomatic-object'. Therefore, the 'Cargomatic-state' is activated as the last activated state in the unloading process. The following explanations regarding 'Cargomatic-stat-function' also apply to the 'Carne-state-function'.

The 'Equipment-variable' is a variable with a variable size as shown in Fig. 15. The 'Cargomatic-state-function' has several inputs including; 'SKU-variable', 'Equipment-variable', 'Door-variable', and time. From the system level, the start and finish time of the 'Cargomatic' determine the start and finish time of the unloading process on input SKU-batches. Therefore, the 'Cargomatic-state-function' updates the operation start and finish time in following variables;

'SKU-variable' and 'Equipment-variable'. The 'Cargomatic-state-function' also changes the value of the 'Availability' in the following variables; 'Equipment-variable' and 'Door-variable'. However, the 'Cargomatic' can perform unloading activity as long as it has enough capacity. The 'Cargomatic-state-function' also updates 'capacity' in 'Equipment-variable' as shown in Fig. 15.

'Cargomatic-state-function' observes the 'Finish' in 'Equipment- variable'. As soon as the 'Finish' was smaller than the simulation time, then the 'availability' of 'Equipment-variable' changes to 'Available' again. Thereafter, the 'Cargomatic-state-function' updates the 'current process' in 'SKU-variable' to P_{12}. This shows the availability of instances of SKU-A for the stacking process.

Stacking and storing process has similar procedures regarding updating the following variables; 'Equipment-variable', 'Infrastructure-variable' and 'SKU-Variable'.

4.4.2 Stored SKUs into shipped orders

When a customer sends an order, a unique instance number is assigned to the coming order according to the instantiation procedure. As a result, order specifications, such as SKU-type, is written in the order dynamic variable as shown in Fig. 16.

The orders require picking and shipping processes. In these two processes, the object-state-functions take into account two main variables; 'order-variable' and 'SKU-variable'. The object-state-functions of operational policies cross-study the registered orders and available SKUs. Accordingly, a pick list is formulated indicating that which SKUs should be picked to satisfy the pick list. Alternative$_A$ employs the 'FIFO-object' to cross-sort SKU and orders instances. First, the orders are sorted by FIFO logic to make the pick-list. Then, the available SKUs are sorted based on the FIFO logic. Subsequently, the 'FIFO-object' writes the sorting results in 'Picking Priority' and 'Priority to being processed', respectively, in 'order-variable' and 'SKU-variable'. The 'FIFO-object' also updates the 'current process' to P_3.

'Forklift-object' is employed as the picking equipment. The 'Forklift-state-function' updates the following values in the order variable; 'Start', 'Finish', and 'Picked status'. The 'Forklift-state-function' also updates the following values in the SKU-variable; Entering-time-to-current process', and 'Finishing-the-process'. 'Forklift-state-function' also changes the 'current process' in SKU-variable to P_4 as soon as the 'Finish' time was elapsed.

The execution principles of the shipping process are similar to the unloading process. The only difference is using 'Forklift-object' instead of 'Cargomatic-object'. However, the 'Sunken-object' updates two values in, respectively, 'SKU-variable and 'order-variable' as soon as orders (picked SKUs) left the warehouse; 'Done with the process flow' and 'order status'. The updates show that the SKUs left the warehouse and the order is dispatched.

4.5 Simulation results

The below given Alternative-Configuration had a lower cost after the application of LP-ALDeP (before simulation) compared to Alternative$_A$ and Alternative$_B$.

$$\begin{aligned}\text{Warehouse}_{\text{Quantitative}-\text{Alternative}_C} = [&(1_1,1),(3_2,\text{NA}),(5_3,1),(7_4,\text{NA}),(7_5,\text{NA}),(10_6,90),(11_7,3),\\&(13_8,\text{NA}),(7_9,\text{NA}),(16_{10},\text{NA}),(11_{11},3),(17_{12},270),(20_{13},\text{NA}),\\&(21_{14},\text{NA}),(15_{15},\text{NA}),(12_{16},3),(6_{17},1),(3_{18},\text{NA}),(8_{19},\text{NA}),(2_{20},3)].\end{aligned}$$

After simulation of Alternative$_A$ and Alternative$_B$, the former demonstrated a lower total cost basically due to lower operational cost.

These three alternatives have the same configurations in unloading, stacking, and shipping processes and they were only different in storing and picking process configurations (employed Design-objects and their quantifications accordingly). Alternative$_A$ and Alternative$_B$ are different in the employed design-objects for the two DRs related to the following operational policies; $\text{Storing} - \text{SKU} - \text{management} - \text{policy} - \text{storage} - \text{allocation}_8$ and $\text{Storing} - \text{Equipment} - \text{management} - \text{policy} - \text{combination} - \text{mode}_{10}$. For the given DRs, respectively, Alternative$_A$ employed 'CBS' and 'Double Command', while Alternative$_B$ employed 'Random-object' and 'Single-command-mode-object'. The difference between Alternative$_A$ and Alternative$_C$ is that the latter employed 'Pallet rack' and 'Turret truck', respectively, for these DRs; $\text{Storing} - \text{Infrastructure} - \text{storage} - \text{module} - \text{type}_{12}$ and $\text{Picking} - \text{Equipment} - \text{type}_{16}$, while Alternative$_A$ employs 'Shelve rack' and 'Forklift'.

The defined KPIs in the SSMM were; fix cost, operational cost, and equipment utilization. The last two KPIs were studied carefully because they are affected by the dynamic behaviour of the design alternatives.

Comparing Alternative$_A$ *and* Alternative$_C$ *from optimization–quantification perspective* 'Pallet rack' is more expensive than the 'Shelve rack', however storing or picking from 'Pallet racks' takes less time, see Appendix A1 in ESM. On the other hand, although 'Turret truck' has shorter operation time (compared to ‘Forklift’) in both storing and picking processes, it is more expensive (fix cost and the required space). LP-ALDeP calculated that in Alternative$_C$ three ‘Forklifts’ is required to perform storing on ‘Pallet rack’ to satisfy the throughput, while in Alternative$_A$ four ‘Forklifts’ are required (since ‘Shelve rack’ is employed). On the other hand, in Alternative$_C$ three ‘Turret truck’ could satisfy the throughput in picking process because ‘Pallet rack’ is employed, while in Alternative$_A$ four ‘Turret truck’ is needed (since ‘Shelve rack’ is employed). Although, ‘Turret truck’ is more expensive, three ‘Turret truck’ cost less than four ‘Forklift’ in the picking process. As a result, the Alternative$_C$ that employs ‘Pallet rack’ and three ‘Forklift’ in storing process and three ‘Turret truck’ in picking process demonstrated less total cost calculated by LP-ALDeP.

Comparing Alternative$_A$ *and* Alternative$_B$ *from the simulation perspective* Alternative$_A$ and Alternative$_B$ employ ‘Shelve rack’ and ‘Forklift’ in storing and picking process. As mentioned, LP-ALDeP calculated that four ‘Forklifts’ were required in each storing and picking processes of these alternatives. Yet, Alternative$_A$ employs ‘CBS’ and ‘Double Command’ as its design-objects for the DRs related to operational policies in storing and picking processes. These two policies contribute to reducing the operational time in storing and picking processes. In particular, CBS allocates SKU-A (as a high demand SKU) to the first four levels of storage modules, while SKU-B and SKU-C are allocated to the fifth levels. As a result, the operation time for SKU-A is reduced, while that is increased for SKU-B and SKU-C. On the other hand, ‘Double command’ policy suggested that after storing the SKUs on storage modules, the equipment can pick the ordered SKUs (from the pick list) and place them in the shipping area. However, this is applicable only if the ordered SKUs are in the same aisle that the SKU storing happened. As a result, in some cases the equipment can cut one empty travelling activity in both storing and picking (without the double command storing equipment could have travelled back empty to the stacking area and the picking could have travelled empty forward to the storing area). Therefore, such a combination of different policies contributes to the reduction in the total operational time in Alternative$_A$. The simulation results of Alternative$_A$ showed that the ‘Forklift’ utilization in the storing process was 59% and 67% in the picking process. In other words, the total working hours of the four ‘Forklift’s in Alternative$_A$ was between the total working hours for two and three ‘Forklift’ in the available time window, as shown below:

$$50\% \left(\frac{2(\text{possible number of forklifts}) \times 5(\text{available time window}) \times 100}{4(\text{existing number of forklifts}) \times 5((\text{available time window}))}\right) < \text{utilization} < 75\% \left(\frac{3 \times 5 \times 100}{4 \times 5}\right).$$

The simulation results of Alternative$_B$ (in terms of ‘Forklift’ utilization) show that it needs all the four Forklifts in both storing and picking processes.

Comparing Alternative$_A$ and Alternative$_C$ from the simulation perspective:

In Alternative$_A$ the number of ‘Forklift’ in both picking and storing processes was reduced to three and it was simulated again. Simulation of the updated Alternative$_A$ revealed that with the new quantification (three Forklifts for each of the storing and picking processes) the throughput was satisfied with a lower total cost compared to Alternative$_C$.

This is an important benefit of applying the proposed framework in this design case. In fact, without addressing the operational policies and simulation, the evaluation of different alternatives (from cost minimization perspective) could lead to losing a better alternative for the detail design. However, the studying warehouse can be considered as small to medium scale size warehouse. Therefore, the aforementioned potential savings can be more significant for a largescale warehouse. Nonetheless, this example is a simple demonstration regarding having an integrated-holistic design approach and the advantageous of early validation (simulation) at early design stages.

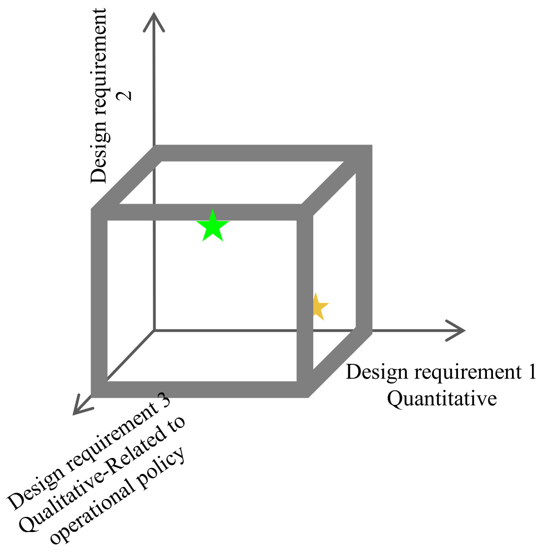


Fig. 17 Multi-dimensional solution space by adding the design requirements related to the operational policy

5 Discussion

The contributions of the introduced modelling approach in this paper can be summarised as follow:

- The paper introduced a full detail approach to develop a system architecture (SSMM) with the following characteristics:
  - Embodies the design knowledge holistically: embodies all types of design requirements from different disciplines, allows the multi-disciplinary and integrated design of CESs.
  - All design alternatives conform to that.
  - Is executable and all design alternatives can be simulated with it.
  - *Is interconnected with the optimization models* A full detail approach was introduced to use optimization models such that leverage the feasible alternatives to a level that can be simulated with the SSMM.
  - *Stands as a design platform* In the early design stages, the system engineer develops the SSMM and simulates a simple behaviour of alternatives to get some insights about the system behaviour. However, by moving forward to the design process (evolution) the detail designers can add (encapsulate) more detail to the SSMM (in particular the object-state-functions). Therefore, the system engineer and detail designers can communicate with the single SSMM, which helps to track the changes and have the same understating about the design process.
- *Reducing the design failure risk* Application of feasibility checking algorithm ensures that the introducing alternatives satisfy the constraints that address the possible inconsistencies between different disciplines.
- *Ensuring data integrity in the framework* All models are systematically interconnected with a set of parameters such that minimises the risk of data mismatch between models that are developed with different participants. This integrity can considerably facilitate the collaborative design of such systems where the design knowledge is complex, massive and multidisciplinary.

Application of the framework in the case study demonstrated certain benefits of that. First, having an integrated design approach opens avenues to find a better design alternative. In the case study, the optimization–quantification models found an optimum solution with regards to the quantifiable DRs without including the DRs related to operational policies. Simulation of alternatives revealed that there is an alternative that can have a lower total cost, which was not identified as the optimum alternative by the optimization–quantification models. In fact, addressing different design disciplines with a holistic approach allows capturing more dimensions of the solution space, which is not limited to the dimensions of the quantifiable DRs. As shown in Fig. 17, an optimization model seeks to find an optimum or near optimum solution in a solution space that its dimension is defined by quantifiable DRs. Nonetheless, when the solution space is searched by adding the dimension of the operational policies, a better solution might be identified. Although some limited aspects of operational policies might be modelled with complicated optimization formulation, the possibility of trapping in local optima might increase by adding more variables to the optimization models. Therefore, this framework could guarantee that its best-identified alternative is at least as good as the best solution that is formulated by optimization–quantification models in the existing literature. On the other hand, it is possible that an optimum result from the optimization–quantification cannot satisfy the initial constraints through simulation. This reference (Wang and Dagli 2013) simulated an optimal solution found by genetic algorithm and showed that the solution could not meet the throughput (that was addressed as optimization constraint). This deviation happened due to the interactions in the system, which the optimization model could not capture it properly, while simulation does.

6 Limitations and future research

Simulation of all the alternatives takes considerable time, and this is a limitation of the framework. Hence, to reduce the computational complexity, application of a supervisory approach can be useful to select a limited number of alternatives for simulation while such selection would not necessarily lead to losing an optimum alternative. However, simulation-based optimization can be used instead of applying them in tandem. However, simulation-based optimisation problems face several challenges, which the computational cost is one of them (Figueira and Almada-Lobo 2014; Steponavičė et al. 2014). In fact, the simulation takes longer time when the number of design variables is large, while the number of decision variables is considerably large in the early design stages. Although, simulation-based optimizations also do not guarantee to find the global optimum, investigation on their application can be future work for this research.

The SSMM simulates the behaviour of an alternative under certain sets of case-specific assumptions regarding the design knowledge, such as deterministic time of machining operations or fixed supply rate of the suppliers. However, the alternative might express a different performance if the assumptions are changed. Therefore, it is useful to address such uncertainties (e.g. stochastic data) in the SSMM to know the magnitude of such a deviation from the initial modelled performance.

7 Conclusion

This paper introduced a framework by application of different modelling methods and formalisms to develop interconnected artefacts, which play certain roles in the framework for the purpose of having integrated and holistic design of complex engineering systems in the early design stages. Accordingly, the framework allows evaluating the impact of each design decisions from different disciplines on system behaviour/performance. With the achieved results it is expectable that this framework may receive attention from designers for real design applications as well as academics for extension and further elaboration.

References

Abdoli S, Kara S (2017a) A modelling framework to design executable logical architecture of engineering systems. Mod Appl Sci 11(9):75

Abdoli S, Kara S (2017b) A review of modelling approaches for conceptual design of complex engineering systems (CESs). In: 2017 IEEE international conference on industrial engineering and engineering management (IEEM). IEEE, Singapore, Singapore

Albers A, Braun A (2011) A generalised framework to compass and to support complex product engineering processes. Int J Prod Dev 15(1–3):6–25

Alfaris AAF (2009) The evolutionary design model (EDM) for the design of complex engineered systems: Masdar City as a case study. Doctoral dissertation, Massachusetts Institute of Technology

Alvarez Cabrera AA, Foeken MJ, Tekin OA, Woestenenk K, Erden MS, De Schutter B, van Tooren MJL, Babuška R, van Houten FJAM, Tomiyama T (2010) Towards automation of control software: a review of challenges in mechatronic design. Mechatronics 20(8):876–886. https://doi.org/10.1016/j.mechatronics.2010.05.003

Alves AC, Silva SC (2009) A review of design methodologies for manufacturing systems. In: 1ST International Conference on Innovations, Recent Trends and Challenges in Mechatronics, Mechanical Engineering and New High-Tech Products Development MECAHITECH‘09. Bucharest

Baker P, Canessa M (2009) Warehouse design: a structured approach. Eur J Oper Res 193(2):425–436. https://doi.org/10.1016/j.ejor.2007.11.045

Bar-Yam Y (2002) General features of complex systems. Encyclopedia of Life Support Systems (EOLSS), UNESCO, EOLSS Publishers, Oxford

Bar-Yam Y (2004) A mathematical theory of strong emergence using multiscale variety. Complexity 9(6):15–24. https://doi.org/10.1002/cplx.20029

Bortolini M, Faccio M, Gamberi M, Manzini R, Pilati F (2016) Stochastic timed Petri nets to dynamically design and simulate industrial production processes. Int J Logist Syst Manag 25(1):20–43

Braha D, Minai AA, Bar-Yam Y (2006) Complex engineered systems: science meets technology. Springer, Berli. https://doi.org/10.1007/3-540-32834-3n

Chakrabarti A, Blessing LTM (2014) An anthology of theories and models of design. Springer, New York

Christophe F, Bernard A, Coatanéa E (2010) RFBS: a model for knowledge representation of conceptual design. CIRP Ann Manuf Technol 59(1):155–158

Cloutier RJ, Verma D (2007) Applying the concept of patterns to systems architecture. Syst Eng 10(2):138–154

Cochran DS, Reinhart G, Linck J, Mauderer M (2000) Decision support for manufacturing system design-combining a decomposition methodology with procedural manufacturing system design. In: The Third world congress on intelligent manufacturing processes and systems. Cambridge

Dauby JP, Dagli CH (2011) The canonical decomposition fuzzy comparative methodology for assessing architectures. IEEE Syst J 5(2):244–255. https://doi.org/10.1109/JSYST.2011.2125250

de Koster R, Le-Duc T, Roodbergen KJ (2007) Design and control of warehouse order picking: a literature review. Eur J Oper Res 182(2):481–501. https://doi.org/10.1016/j.ejor.2006.07.009

Dori D (2002) Object-process methodology. Springer, New York

Dori D, Renick Aharon, Wengrowicz Niva (2016) When quantitative meets qualitative: enhancing OPM conceptual systems modeling with MATLAB computational capabilities. Res Eng Design 27(2):141–164. https://doi.org/10.1007/s00163-015-0209-9

Douglass BP (2016) Chapter 1—What is model-based systems engineering? Agile Systems Engineering. Morgan Kaufmann, Boston, pp 1–39

ElMaraghy HA, Kuzgunkaya O, Urbanic RJ (2005) Manufacturing systems configuration complexity. CIRP Ann Manuf Technol 54(1):445–450. https://doi.org/10.1016/S0007-8506(07)60141-3

Estefan JA (2003) Survey of model-based systems engineering (MBSE) methodologies. Jet Propulsion Laboratory, Pasadena

Figueira G, Almada-Lobo B (2014) Hybrid simulation–optimization methods: a taxonomy and discussion. Simul Model Pract Theory 46:118–134

Fishwick PA (2007) Handbook of dynamic system modeling. CRC Press, Boca Raton

Fleck M, Berardinelli L, Langer P, Mayerhofer T, Cortellessa V (2013) Resource contention analysis of service-based systems through fUML-driven model execution. Proc. of NiM-ALP:6

Gausemeier J, Dumitrescu R, Kahl S, Nordsiek D (2011) Integrative development of product and production system for mechatronic products. Robot Comput Integr Manuf 27(4):772–778. https://doi.org/10.1016/j.rcim.2011.02.005

Gu P, Rao HA, Tseng MM (2001) Systematic design of manufacturing systems based on axiomatic design approach. CIRP Ann Manuf Technol 50(1):299–304

Hopp WJ, Spearman ML (2011) Factory physics. Waveland Press, Long Grove

Huang E, Ramamurthy R, McGinnis LF (2007) System and simulation modeling using SysML. In: Proceedings of the 2007 winter simulation conference. IEEE, Washington, DC, USA

INCOSE (2015) INCOSE SE Handbook Working Group, INCOSE system engineering handbook San Diego, 4th edn. Wiley, Hoboken

Jørgensen HD (2004) Interactive process models. Doctoral thesis, Norwegian University of Science and Technology, Fakultet for informasjonsteknologi, matematikk og elektroteknikk

Kapos GD (2015) Enabling system models automated evaluation through cross-concept information utilization. In: 2015 IEEE 9th international conference on research challenges in information science. IEEE, Athens, Greece

Kapos GD, Dalakas V, Nikolaidou M, Anagnostopoulos D (2014) An integrated framework for automated simulation of SysML models using DEVS. Simulation 90(6):717–744

Khan I (2010) Methodology for the development of executable system architecture. In: Proceedings of the 8th international conference on frontiers of information technology. ACM, Islamabad. https://doi.org/10.1145/1943628.1943677

Komoto H, Tomiyama T (2012) A framework for computer-aided conceptual design and its application to system architecting of mechatronics products. Comput Aided Des 44(10):931–946. https://doi.org/10.1016/j.cad.2012.02.004

Koo HYB (2005) A meta-language for systems architecting. Technion, Israel Institute of Technology, Haifa

Kossiakoff A, Sweet WN, Seymour SJ, Biemer SM (2011) Systems engineering principles and practice, vol 83. Wiley, New York

Maropoulos PG, Ceglarek D (2010) Design verification and validation in product lifecycle. CIRP Ann 59(2):740–759. https://doi.org/10.1016/j.cirp.2010.05.005

Matei I, Bock C (2012) SysML extension for dynamical system simulation tools. US Department of Commerce, National Institute of Standards and Technology. https://doi.org/10.6028/NIST.IR.7888

Mayerhofer T, Langer P, Wimmer M, Kappel G (2013) xMOF: executable DSMLs based on fUML. In: International conference on software language engineering, Springer, Cham, pp 56–75. https://doi.org/10.1007/978-3-319-02654-1-4

McGinnis LF, Ustun V (2009) A simple example of SysML-driven simulation. In: Proceedings of the 2009 winter simulation conference. Austin, Texas, pp 1703–1710

McGinnis LF, Huang E, Wu K (2006) Systems engineering and design of high-tech factories. In: Proceedings of the 2006 winter simulation conference. IEEE, Monterey, CA, USA. https://doi.org/10.1109/WSC.2006.322969

McGinnis L, Huang E, Kwon KS, Ustun V (2011) Ontologies and simulation: a practical approach. J Simul 5(3):190–201

Meng X (2010) Modeling of reconfigurable manufacturing systems based on colored timed object-oriented Petri nets. J Manuf Syst 29(2–3):81–90. https://doi.org/10.1016/j.jmsy.2010.11.002

Mijatov S, Mayerhofer T, Langer P, Kappel G (2015) Testing functional requirements in UML activity diagrams. International conference on tests and proofs. Part of the Lecture notes in computer science, LNCS, vol 9154. Springer, Cham, pp 173–190. https://doi.org/10.1007/978-3-319-21215-9_11

Mönch L, Lendermann P, McGinnis LF, Schirrmann A (2011) A survey of challenges in modelling and decision-making for discrete event logistics systems. Comput Ind 62(6):557–567. https://doi.org/10.1016/j.compind.2011.05.001

Moses J (2002) The anatomy of large scale systems. ESD Internal Symposium

Nikolaidou M, Kapos GD, Dalakas V, Anagnostopoulos D (2012) Basic guidelines for simulating SysML models: an experience report. In: 2012 7th international conference on system of systems engineering (SoSE), IEEE, Genova, Italy, pp 95–100. https://doi.org/10.1109/SYSoSE.2012.6384172

Osorio CA, Dori D, Sussman J (2011) COIM: an object-process based method for analyzing architectures of complex, interconnected, large-scale socio-technical systems. Syst Eng 14(4):364–382

Pahl G, Beitz W, Feldhusen J, Grote KH (2007) Engineering design—a systematic approach, 3rd edn. Springer, New York

Pape L, Giammarco K, Colombi J, Dagli C, Kilicay-Ergin N, Rebovich G (2013) A fuzzy evaluation method for system of systems meta-architectures. Procedia Comput Sci 16:245–254. https://doi.org/10.1016/j.procs.2013.01.026

Reich Y (2017) What is a reference? Res Eng Design 28(4):411–419. https://doi.org/10.1007/s00163-017-0270-7

Robinson S (2006) Conceptual modeling for simulation: issues and research requirements. In: Proceedings of the 2006 winter simulation conference. IEEE, Monterey, CA, USA, pp 792–800. https://doi.org/10.1109/WSC.2006.323160

Rouwenhorst B, Reuter B, Stockrahm V, van Houtum GJ, Mantel RJ, Zijm WHM (2000) Warehouse design and control: framework and literature review. Eur J Oper Res 122(3):515–533. https://doi.org/10.1016/S0377-2217(99)00020-X

Roy R, Hinduja S, Teti R (2008) Recent advances in engineering design optimisation: challenges and future trends. CIRP Ann Manuf Technol 57(2):697–715. https://doi.org/10.1016/j.cirp.2008.09.007

Schönherr O, Rose O (2009) First steps towards a general SysML model for discrete processes in production systems. In: Proceedings of the 2009 Winter Simulation Conference (WSC). IEEE, Austin, TX, USA, USA, pp 1711–1718. https://doi.org/10.1109/WSC.2009.5429164

Schotborgh WO, McMahon C, Van Houten FJAM (2012) A knowledge acquisition method to model parametric engineering design processes. Int J Comput Aided Eng Technol 4(4):373–391

Schuh G, Monostori L, Csáji BC, Döring S (2008) Complexity-based modeling of reconfigurable collaborations in production industry. CIRP Ann Manuf Technol 57(1):445–450

Sitton M, Reich Y (2018) EPIC framework for enterprise processes integrative collaboration. Syst Eng 21(1):30–46

Steponavičė I, Ruuska S, Miettinen K (2014) A solution process for simulation-based multiobjective design optimization with an application in the paper industry. Comput Aided Des 47:45–58

Thiers G (2014) A model-based systems engineering methodology to make engineering analysis of discrete-event logistics systems more cost-accessible. Doctoral dissertation, Georgia Institute of Technology

Tomiyama T, D'Amelio V, Urbanic J, ElMaraghy W (2007) Complexity of multi-disciplinary design. CIRP Ann Manuf Technol 56(1):185–188

Tomiyama T, Gu P, Jin Y, Lutters D, Kind C, Kimura F (2009) Design methodologies: industrial and educational applications. CIRP Ann Manuf Technol 58(2):543–565. https://doi.org/10.1016/j.cirp.2009.09.003

Umeda Y, Ishii M, Yoshioka M, Shimomura Y, Tomiyama T (1996) Supporting conceptual design based on the function-behavior-state modeler. Artif Intell Eng Des Anal Manuf 10(04):275–288

Vrabič R, Butala P (2011) Computational mechanics approach to managing complexity in manufacturing systems. CIRP Ann Manuf Technol 60(1):503–506. https://doi.org/10.1016/j.cirp.2011.03.050

Wagenhals LW, Haider S, Levis AH (2003) Synthesizing executable models of object oriented architectures. Syst Eng 6(4):266–300. https://doi.org/10.1002/sys.10049

Wang R (2012) Search-based system architecture development using a holistic modeling approach. Doctoral Dissertation, Missouri University of Science and Technology (2256)

Wang R, Dagli CH (2008) An executable system architecture approach to discrete events system modeling using SysML in conjunction with colored Petri Net. In: 2008 2nd annual IEEE systems conference. IEEE, Montreal, Que., Canada, pp 1–8. https://doi.org/10.1109/SYSTEMS.2008.4518997

Wang RZ, and Dagli CH (2013) Developing a holistic modeling approach for search-based system architecting. In: 2013 Conference on Systems Engineering Research, vol 16, pp 206–215. https://doi.org/10.1016/j.procs.2013.01.022

White BE (2007) On interpreting scale (or view) and emergence in complex systems engineering. In: 2007 1st annual IEEE systems conference. IEEE, Honolulu, HI, USA, pp 1–7. https://doi.org/10.1109/SYSTEMS.2007.374660

Wolfram S (1985) Complex systems theory. The Institute for Advanced Study, Princeton

Yaroker Y, Perelman V, Dori D (2013) An OPM conceptual model-based executable simulation environment: implementation and evaluation. Syst Eng 16(4):381–390

Zheng C, Hehenberger P, Duigou JL, Bricogne M, Eynard B (2016) Multidisciplinary design methodology for mechatronic systems based on interface model. Res Eng Des 28(3):1–24

Zhow W, Yang F, Zhu Y (2015) A transformation method of OPM Model to CPN model for system concept development. In: Proceedings of the First International Conference on Information Science and Electronic Technology (ISET)

Ziv-Av A, Reich Y (2005) SOS–subjective objective system for generating optimal product concepts. Des Stud 26(5):509–533