

A Modelling Framework to Design Executable Logical Architecture of Engineering Systems

Shiva Abdoli¹ & Sami Kara¹

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

Correspondence: Shiva Abdoli, School of Mechanical and Manufacturing Engineering, University of New South Wales, Sydney, Australia. E-mail: s.abdoli@student.unsw.edu.au

Received: July 27, 2017

Accepted: August 4, 2017

Online Published: August 24, 2017

doi:10.5539/mas.v11n9p75

URL: <https://doi.org/10.5539/mas.v11n9p75>

Abstract

Modern production, logistic and assembly systems comprise of a considerable number of processes which operate by using diverse types of resources. Conceptual design of these systems has become more complicated because of the large scale and multi-disciplinary essence of their design knowledge. This paper proposes a modelling framework to support the conceptual design of such systems. The framework employs the principles of system engineering to fulfil the necessity of having a multi-disciplinary approach for the design of such systems. The framework realizes the essence of holistic design by modelling the structural and behavioral aspects of a system in one design artefact. Object Oriented (OO) method is employed to facilitate the complexity of holistic analysis and yielding proper logics for system architecting. This paper proposes incorporating the OO analyzing semantics into Finite State Machine formalism. Therefore, the logical architecture will be established in an FSM platform. In return, the resultant artefact can stand as an executable Meta-Architecture such that design alternatives are its instances. Moreover, the Meta-Architecture enables simulation of the alternatives which serve their early validation. Accordingly, this approach opens avenues regarding incorporation of the Meta- Architecture with computational and analyzing methods which can significantly support the decision making in the conceptual design stage.

Keywords: engineering systems, object oriented method, system engineering, system logical architecture, discrete event simulation, finite state machine

1. Introduction

Manufacturing, production, assembly, logistic and warehousing systems are the main players in supply chains of widespread industries such as food, chemical, fabrication, wood, and electronics. These systems mainly operate based on certain processes that interacting to fulfil the overall system functionality.

The high pace of market evolution puts more pressure on businesses to shorten their design time. This highlights the importance of further studying and researching regarding the design practices of these systems.

Conceptual design has been proven to have the highest impact on life cycle performance, costs and environmental impacts of the aforementioned systems (Christophe, Bernard, & Coatanéa, 2010; P. Gu, Rao, & Tseng, 2001; Umeda, Ishii, Yoshioka, Shimomura, & Tomiyama, 1996). Hence, this paper focuses on their conceptual design stage. In the following section first the characteristics of such systems are described from the design perspective. Then, the research aims and scope are clarified.

1.1 Characteristics of Targeted Systems from the Design Perspective

This paper targeted those systems that fulfil their required functionally by operating based on certain independent processes. They interact dynamically for the purpose of item transition through the system. However, internal operations of the processes are independent.

The overall system behavior in terms of item transition can be characterized with some events happening at discrete time points. Hence, the dynamic behavior of such systems is considered discrete in the system level (L. F. McGinnis, E. Huang, & K. Wu, 2006; Mönch, Lendermann, McGinnis, & Schirrmann, 2011).

Each process may operate by using diverse types of resources. The processes require a set of resources (soft and

or physical) for their operations. Hence, it is common that the design of such systems requires involvement of multiple design disciplines (Zheng, Hehenberger, Le Duigou, Bricogne, & Eynard, 2016).

The explained characteristics can be translated into three main attributes; large scale, multi-disciplinary and dynamic behavior. This structure lends these systems to complex system class from the design perspective (Bar-Yam, 2002; ElMaraghy, Kuzgunkaya, & Urbanic, 2005; Koo, 2005; Schuh, Monostori, Csáji, & Döring, 2008; Wolfram, 1985). Hence, the targeted systems can be called ‘Complex Engineering Systems’ (CESs). In this paper, CES is an umbrella term to facilitate referring to the aforementioned systems.

1.2 Research Scope

Managing the complexity of the design knowledge is one of the most challenging issues in the conceptual design of complex systems (ElMaraghy et al., 2005; SE Handbook Working Group, 2011). Particularly, it is challenging to represent and structure the design knowledge in a proper artefact. In the design of CESs, the scale of the system made the designers from the different disciplines to design these systems in silos (Baker & Canessa, 2009; Fishwick, 2007; Mönch et al., 2011; Rouwenhorst et al., 2000; Wang, 2012). However, it is likely that some aspects of the system and its design knowledge be overlooked (Tetsuo Tomiyama, D’Amelio, Urbanic, & ElMaraghy, 2007). This may lead to inconsistencies between different disciplines and eventually design failures which prolong the design process.

On the other hand, the existing design approaches mostly follow a top-down approach (Mönch et al., 2011; Pape et al., 2013; Wang, 2012). In general, following the top-down approach may lead to narrowing down the solution space. However, overlooking some aspects of the design solution may lead to losing a better solution and a huge loss. In particular, operational policies or non-physical aspects are mostly designed when physical design is finalized. Physical design here refers to equipment selection or infrastructure determination. Nonetheless, the CESs mainly achieve their intended overall functionality through proper interaction of their processes. Hence, it is crucial to observe the impact of the design decisions on dynamic behavior of the system in order to make more realistic decisions in conceptual design stage. These issues emphasize on the importance of having a holistic approach in terms of designing both operational policies and physical aspects of a system simultaneously.

In the engineering design, models are conceptual artefacts (ANDREASEN, 1994). A proper modelling approach can make the solution more transparent (Simon, 1996). To the best of the author’s knowledge, researches that aimed on developing generic modelling approaches that can assist on holistic design of CESs are so scarce or limited (Leon F McGinnis, Edward Huang, & Kan Wu, 2006; Mönch et al., 2011; Thiers, 2014; T. Tomiyama et al., 2009; Wang & Dagli, 2008). This paper is part of an ongoing research which aims at proposing an integrated modelling framework that facilitates the holistic design of CESs by proper application of its employed methods. Nonetheless, the core of modelling approach is explained in this paper. Hence, this paper aims at proposing a modeling approach that enables capturing the design knowledge holistically by accomplishing two objectives;

- a. Demonstrating the system structure holistically
- b. Addressing the dynamic behavior in the system structure

Model-based design is a promising approach to cope with the growing complexity of systems (Grobshtein, Perelman, Safra, & Dori, 2007). On the other hand, Model-based design is a way of formalization. Formalization aims at defining the semantics of representing something before describing that. Thus, Model-based design can assure relaxing the application of the employed methods in the proposing framework from the CES domain (Thiers, 2014).

2. Literature Review

Several researches have classified the existing design concepts from different perspectives (Buede & Miller, 2016; McGinnis, Huang, Kwon, & Ustun, 2011). Engineering design methodologies describe the design with a logical sequence of tasks. Yet, design methodologies have been less applied in industry due to several reasons (Alves & Silva, 2009; T. Tomiyama et al., 2009). For example, design methodologies mostly guide the design and do not search for or generate solutions. However, models can envelope specifications of a specific design case. Hence, models can contribute to setting up a design solution (Coatanéa et al., 2015). Modelling framework is defined as a logical structure that can be applied in design practices to classify the design knowledge or to provide mechanisms for analyzing the information (R. J. Cloutier & Verma, 2007). Modelling frameworks can support the design process in two ways; first, defining what the structure of the design is in terms of required tasks and their sequence, and second, how to perform those tasks in terms of employing proper approaches. The former class is called descriptive and the latter is called prescriptive approaches (Andersen, Brunoe, Nielsen, & Rösiö, 2017; Tetsuo Tomiyama, 2006). However, the scope of this paper is aligned with the prescriptive

modelling framework concept. Yet, the most used modeling languages are also reviewed briefly (in the context of conceptual design stage).

2.1 Modelling Languages

Unified Modelling Language (UML) is extensively used in Object Oriented (OO) design of software systems. However, UML got attentions in other domains as well (Koo, 2005; McGinnis et al., 2011; Wang, 2012). System modelling language (SysML) is an extension of UML. Both languages can help to visualize the design knowledge in terms of representing the system structure in early stages of the design (Barbieri, Fantuzzi, & Borsari, 2014; Penas, Plateaux, Patalano, & Hammadi). Yet, both have serious flaws on executing the dynamic behavior of the system (Yaroker, Perelman, & Dori, 2013).

A Petri Net (PN) model is a bi-partite graph encompassing places, transitions and arcs (Zhu & Brooks, 2009). A PN model can capture the dynamic state of a system. Colored PN (CPN) is an extension of PN. In a CPN model, the tokens can carry certain properties.

Object Process Methodology (OPM) is a modelling language with a diagrammatic view (Dori, 2002). OPM models a system based on objects, processes and links. OPM incorporates certain architecting and execution semantics. Yet, OPM has flaws regarding specifying a formal computational model for either discrete or continuous event systems. Moreover, OPM is less scalable to include descriptive details in comparison to UML (Koo, 2005; Wang, 2012).

Foundational Subset for Executable UML Models (FUML) contains certain behavioral semantics. FUML models require a virtual machine and trace models to simulate sequential runtime behaviors. Yet, certain concepts such as event or constraint are not defined in the existing reference implementation of FUML. A dedicated trace model is also required for any execution scenario. These flaws limit FUML capabilities for validation of the dynamic aspects of the CESs (Mayerhofer, Langer, Wimmer, & Kappel, 2013; Mijatov, Mayerhofer, Langer, & Kappel, 2015).

2.2 Methods for Structuring the Design Knowledge

In the conceptual design stage, structuring of the design knowledge refers to forming the overall topology of a system by defining its specifications. Reviewed approaches here may overlap, because, each approach looks at the system from a different perspective sharing some common basics.

2.2.1 Systems Engineering (SE)

SE is an interdisciplinary discipline that focuses on the design of the whole system as distinct from its elements (SE Handbook Working Group, 2011). There are several SE models that mainly differ on execution sequence of SE tasks. Nonetheless, SE tasks can be enveloped to four main tasks; requirement analysis, system architecture design, detail design, validation and verification as shown in rectangles in upper section of Figure.1 (Kossiakoff, Sweet, Seymour, & Biemer, 2011). Initially, informal customer requirements are translated into system requirements. Then, system engineer designs the system logical architecture in terms of its decomposition to logical subsystems, their system level requirements and subsystems interaction interface (Douglass, 2016; Osorio, Dori, & Sussman, 2011). Conceptual design is considered equivalent to designing the system logical architecture (Komoto & Tomiyama, 2012). The system engineer can develop some design alternatives by selecting from possible key options for the requirements without focusing on the detail design. The system engineer studies the performance of alternatives, and accordingly nominates a preferred candidate(s) for the detail design. This analysis is briefly termed trade study. Down-stream engineers develop the detailed designs of subsystems for the nominated alternatives. Design alternative(s) need to be validated and verified with respect to the requirements. Model Based System Engineering (MBSE) is an extension of SE. MBSE enriches SE with models application (Friedenthal, Meilich, & Izumi, 2007). MBSE uses models as a baseline for the associated SE tasks (Bergenthal, 2011).

2.2.2 Functional Modelling (FM)

FM aims at describing how a system goal is achieved by realizing its functions. Different ontologies exist for FM such as; Device centric (DC) and Function-behavior-state (FBS). DC treats a system as a composition of black boxes connecting with inputs-outputs. DC hardly considers the effect of an object on its environment. FBS defines a function as a subjective description of a behavior. Behaviors are objectively formulated in different models belonging to different design disciplines. FM has been mostly practiced in mechanical product design. Hence, FM mostly focuses on modelling the physical aspects of a system and overlooks the soft design requirement of the system of interest (Erden et al., 2008). Moreover, FM does not give equal attention to the system and its environment.

2.2.3 Object Oriented (OO)

OO modelling aims at modelling a system with its objects (Bennett, McRobb, & Farmer, 2005; Booch, 1994). Objects characteristics reflect the knowledge of the system about its objects. A class describes a set of objects that share specific characteristics. An object is a particular instance of a class. Abstraction and generalization are two OO principles. Generalization is a kind of taxonomic relationship among certain classes. Generalization means any instance of a more specific class is an instance of a more general class. Classes can be defined abstract or concrete. Abstract classes cannot have their own direct objects. Classes can be derived from an abstract class and are called derived or sub-classes of the abstract class. Use cases describe the system usage scenarios from the user's perspective.

2.3 Modelling Framework

In the context of system design, a prescriptive modelling framework clarifies the design tasks in terms of; what to do, how to do and when to do them (Albers & Braun, 2011; Estefan, 2003; T. Tomiyama et al., 2009). Hence, descriptive frameworks are excluded from the review such as Object-Oriented SE Method (Friedenthal et al., 2007) or Department of Defense Architecting Framework (Jepsen, Mortensen, & Hvam, 2014).

Reference (Wang, 2012) introduced a modelling framework for system architecting that combines the capabilities of OPM, CPN and 'feature model'. The framework uses 'feature model' to represent the system structure. CPN is utilized as the target modelling formalism to simulate the behavior of the system. OPM is utilized to serve as a hub between those models. Hence, certain CPN semantics were incorporated into the OPM model to support the 'Model to Model' transformation (M2M); from OPM to CPN formalism. Genetic Algorithm was used to search the solution space. It was suggested to transform the CPN model to chromosome encoding for the purpose of searching the solution space. Yet, the framework does not suggest any explicit guideline regarding modelling the system structure. In fact, it was assumed that the system structure will be an input to the framework. Moreover, it was assumed that the embodiment designs of requirements are given as the building blocks of the system architecture. Hence, the described architecting approach basically plays as a configuration mechanism. Moreover, that framework requires the integration of multiple formalisms with several M2M transformations which is fairly complicated. However, due to technical complexity, the research could not achieve fully automatic M2M transformations.

Reference (Thiers, 2014) introduced a methodology to support the design and analysis of logistic systems by applying of MBSE. The methodology suggested using SysML for developing a descriptive user model. The methodology suggested building the analysis models from the descriptive model using an automated builder program. An abstraction of token flow network was defined to facilitate the M2M transformation from SysML to CPN. However, as it is stated in that research, such automatic builder does not exist and its development has been considered as a future research.

References (Bienvenu, Shin, & Levis, 2000; R. Cloutier & Griego, 2008) aimed on developing a rationale for modelling the system architecture by application of OO method. Both works targeted systems that were related to Department of Defense missions. Obviously, the scope of those works is different from this paper and the developed rationale was dependent on that study domain. However, both works did not introduce clear architecting guidelines. Indeed, those publications mainly suggested using multiple graphs of UML/SysMI for presenting the system structure (such as class diagram, sequence and block diagram).

Accordingly, three specific shortcomings have been identified in the reviewed literatures which are summarized below and are described afterwards;

1-Lack of prescribing architecting guidelines

2-Demonstrating the system structure and realizing the dynamic behavior in separated modelling formalisms

3-Modelling the operational policies as the model invariants

The existing frameworks mostly sufficed on suggesting a modelling approach and relied on the designer experience for system architecting. However, modelling is both art and science. Two designers may use one modelling approach and the resultant models can be different. Thus, a framework should provide certain perspective guidelines for the system architecting. Hence there is a lack in literature regarding proposing prescribing guidelines for modelling the logical architecture of CESs.

On the other hand, the existing frameworks mostly present the logical architecture with the descriptive/static modelling languages. However, the resulted artefacts have shortcomings regarding capturing the dynamic behavior of the system (in particular simulation) (Matei & Bock, 2012). Current approaches seek to bridge this

gap by using complex chain of M2M transformation (Fleck, Berardinelli, Langer, Mayerhofer, & Cortellessa, 2013). However, developing and working with such a transformer requires high level of expertise in programming. Yet, such a transformer does not completely exist and still evolving (Wang & Dagli, 2008). Hence, these approaches have some shortcomings regarding automatically driving a simulation model from the descriptive model (Alvarez Cabrera et al., 2010; Fleck et al., 2013; Matei & Bock, 2012; Mayerhofer et al., 2013; Mijatov et al., 2015; Moses, 2002).

To the best of the authors’ knowledge, scholarly frameworks develop simulation models so that the operational policies of a system or its control procedures are modelled as a fixed part of the model. As a result, generation and execution of different simulation scenarios do not include changes in types of operational policies or procedures. However, they can be critical design requirements and strongly affect the dynamic behavior of the system. For example, reference (Thiers, 2014) suggested modelling the system procedures in behavioral diagrams of SysML. Yet, this approach dictates the operational procedures to the generated design alternatives (Thiers, 2014). Similarly, the developed model by reference (Wang, 2012) cannot generate alternatives which vary in their operational policies.

As conclusion, existing frameworks have flaws regarding capturing the design knowledge holistically in terms of addressing both structural and behavioral aspects in one model. This research aims to overcome these shortcomings by incorporating the excelling capabilities of existing design methods and modelling formalisms.

3. Framework Structure

Recall the given definition for the modelling framework. Thus, the first step should be determining the structure of the proposing framework in terms of the required design tasks and their logical sequence. Thereafter, application of proper methods is prescribed for performing those tasks.

3.1 Framework Structure

SE is a well-known approach to assist the design of multidisciplinary systems. SE encompasses interdisciplinary activities and concentrates on system properties rather than on specific requirements for single disciplines (Kossiakoff et al., 2011)

Hence, the proposed framework will be structured by using the SE principles.

Generally, business experts analyze a system within its domain and provide the corresponding remarks for the designers. On the other hand, detail designs are basically domain and discipline dependent. Hence, this research assumes that the design requirements are provided and the downstream engineers take care of the detailed design. Thus, this research focuses on designing the CES logical architecture and validating of the design alternatives at the system level. Therefore, the required tasks in the proposing framework is given below and shown as highlighted rectangles in the upper section of Figure. 1.

- I. Designing the system logical architecture
- II. Validation of design alternatives

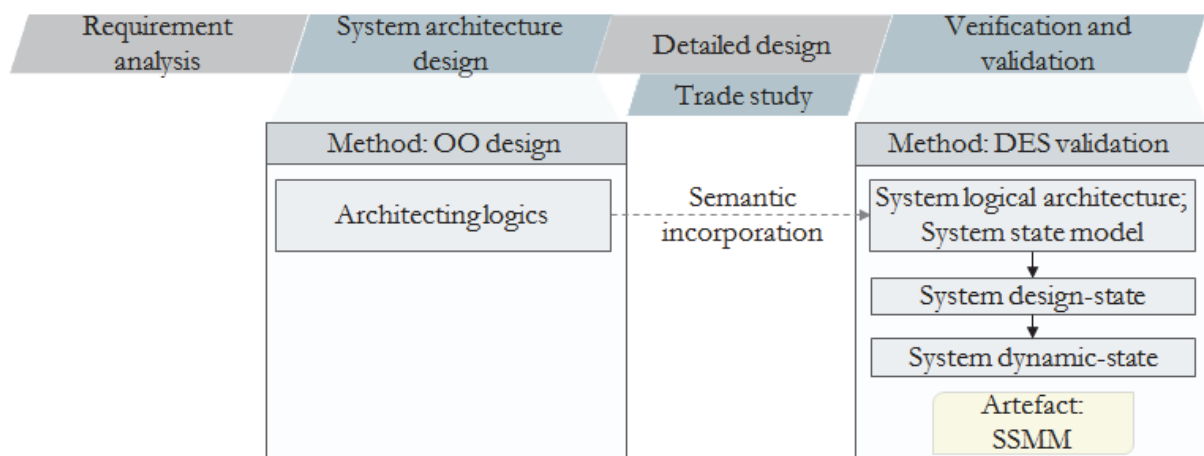


Figure 1. Modelling framework tasks, methods and their artefacts in conjunction with SE process flow

As discussed in the introduction, this research aims at capturing the design knowledge holistically by accomplishing two objectives; a) demonstrating the system structure holistically, b) addressing the dynamic behavior in the system structure. Obviously, the former objective (a. demonstrating the system structure holistically) can be addressed in the first task of the framework (I. Designing the system logical architecture). In fact, the framework should propose certain architecting guidelines to develop the system logical architecture. The later objective (b. addressing the dynamic behavior in the system structure) can be addressed in the second task of the proposing framework (II. validation of design alternatives). To this end, the logical architecture should also embody the dynamic essence of the CES. Hence, the architecting logics should enable addressing both the operational policies and the physical aspects in the system logical architecture. In this context, validation should enable observing the dynamic essence of the system. Thus, the logical architecture artefact should have the potentials to be executed later. This satisfies the purpose of observing the impact of design decisions (including both physical design and operational policy design) on the system dynamic behavior.

3.2 Method for System Logical Architecting

Generally, complex systems consist of multiple hierarchies. OO modelling is a well-known approach to design the complex software systems, especially where the systems do not have a single hierarchy. Hence, OO method can assist on capturing the complexity of the holistic design (Acheson, 2010). This research proposes application of OO method for designing the CES logical architecture as shown in Figure. 1. In particular, OO analysis principles are employed to achieve proper architecting logics for holistic modelling of the system structure. In return, the elaborated logical architecture can semantically stand as a Meta-Architecture of the CES such that the design alternatives are instances of the Meta-Architecture. Indeed, the resultant architecting guidelines can serve the first identified gap in this research context (1-lack of architecting guidelines).

3.3 Method for Validation of Dynamic Behavior

Discrete event simulation (DES) has proved to be a good method for validation of the dynamic aspect of CESs (Wang & Dagli, 2008). DES approach tries to model a system as such the events happen in a chronological sequence (Robinson, 2014). Four main discrete-event modelling formalisms are defined; process-oriented, activity-oriented, event-oriented, and state-oriented as shown in Figure. 2 (Fishwick, 2007; Miller, Baramidze, Sheth, & Fishwick, 2004; Zhu & Brooks, 2009).

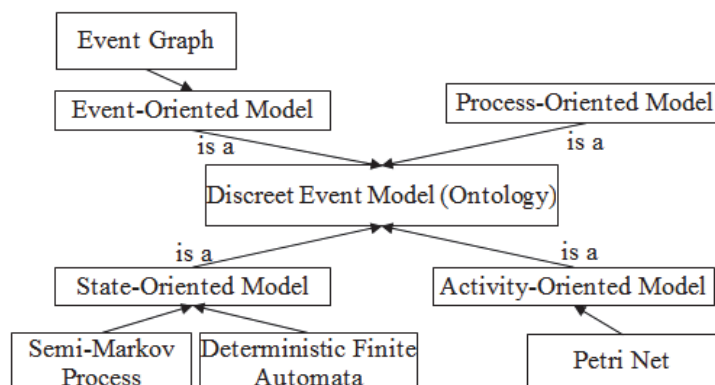


Figure 2. Paradigms of discrete event modelling

The commercial simulation platforms usually operate based on one of these formalisms. These platforms provide a modelling environment which simplifies the modelling task. Yet, this simplicity comes at the cost of reducing the model flexibility. However, the model flexibility is crucial in the conceptual modelling stage (Meta-Architecting). Hence, it is better to using one of the four DES formalisms for modelling the dynamic behavior of the CESs.

In an event graph, nodes represent the events and edges demonstrate causality. Process-Oriented ontology models a system by means of processes, their states and transition functions. Finite State Machine (FSM) formalism is a generic form of state oriented approach. A FSM model contains the possible states that a system can face and the rules of transition between states. PN formalism has been described earlier. 'PN models' and 'event graphs' are network models. However, the model structure can look like fairly complicated by using the network models, even for small scale systems (Wagenhals, Haider, & Levis, 2003; Yaroker et al., 2013) Moreover, networks mainly show the causality by arcs. Hence, these models cannot simply show certain aspects

of the system structure such as hierarchy or commonality. Hence, these network modelling formalisms have flaws regarding expressing the system structure simply/meaningfully. Obviously, process oriented ontology neglects structural aspects of a system and mainly focuses on its processes. However, a system is more than of its events and processes. In fact, a system has a structure along with its behavioral mechanism, which both together lead to certain events in the system.

Nonetheless, the aforementioned formalisms overlap on some common basics. However, among the aforementioned formalisms, FSM models can present the system structure more meaningfully. In fact, FSM firstly models a system structure based on its elements. Then, the system structure can be enhanced by defining certain dynamic states for the system elements. These dynamic states indicate the behavior of the system elements from DES perspective. Hence, FSM can represent the system structure as well as its dynamic state. This may ease FSM usability for the purpose of modelling the system logical architecture in conceptual design stage. Thus, this research proposes establishing the Meta-Architecture in an FSM platform. To this end, the framework incorporates the utilized OO semantics into FSM formalism. As a result, the elaborated Meta-Architecture has the potentials to be DES executable as shown in Figure. 1. In return, the structural and behavioral aspects of a system can be demonstrated and examined holistically in one model. In fact, this approach can fill the second identified gap in this research context (2- demonstrating the system structure and realizing the dynamic behavior in separated modelling formalisms). The promoted Meta-Architecture is called System-State-Meta-Model (SSMM).

It will be explained in detail later that how the proposing framework serves the third research gap (3- Modelling the operational policies as the model invariants). However, the main idea relies on modelling a CES as a machine (SSMM). Accordingly, design requirements are considered equivalent to the states of the machine (SSMM) with respect to its structure (design-requirement-state). Each design requirement state includes certain sub-states representing its possible design options (object-states). Therefore, selection of a key option for a single design requirement (including operational policies) is equivalent to activating one object-state. Thus, a design alternative can be developed by activating one object-state for each and all of the design-requirement-states. Hence, the operational policies are not dictated to the model (SSMM). Indeed, the design alternatives can vary in their selected operational procedures.

SSMM can provide a formal communication framework between SE phases. Although, the detail design task is not targeted in this framework, yet, the SSMM indicates the design requirements from the down-stream engineers, let's call them design modules. SSMM enables encapsulation of the detailed designs into those modules. Nonetheless, the system engineer can also examine the dynamic behavior of the design alternatives in a simplified form even before the detail design. In fact, the detail design can solidify the validation results.

The CESs exist in widespread domains. Generally, domain models should present a clear picture of domain axioms without any information about a specific system. Domain models are developed based on certain intrinsic and extrinsic assumptions that are basically structural facts or behavioral laws of that domain. Structural facts are fundamental constants of a domain, whereas behavioral laws demonstrate the reactive nature of the domain (Mashkoor & Jacquot, 2011). This research aims at formalizing the approach of addressing the structural facts and behavioral laws in the emergent artefacts of the framework. Therefore, the framework will be domain neutral. Generally, the logical architecture of a system is defined as composition of a set of concepts and principles supporting the logical operation of the system (SEBoK authors, 2016). Thus, the domain structural facts will be addressed in the first task of the framework (designing the logical architecture). As mentioned, this framework uses DES for the validation task. Obviously, DES investigates the functionality of the design by examining the dynamic behavior of the system. Hence, the behavioral laws will be addressed in the second task of the framework.

The next two sections discuss about proper implementation of the aforementioned methods in the proposed framework. A warehouse system is used as a CES example to demonstrate the applicability of the framework. The main artefact of the framework (SSMM) is implemented in Matlab state flow library. However, the explained semantics can be implemented in other similar platforms as well.

4. Methods Application

4.1 CES Logical Architecting by Application of OO

It is worth articulating that the system logical architecture in the proposed framework should clearly express the allocation of the design requirements into the decomposed sub-systems. Generally, the OO design of a system comprises couple of tasks; I- system abstraction II- use case development III- Specific design establishment.

UML is one of the most used formal modelling languages in OO design applications (Bennett et al., 2005; Booch, 1994; Koo, 2005; McGinnis et al., 2011; Wang, 2012). Thus, this research describes its suggesting architecting guidelines by using the UML semantics. However, the novelty in proposing the architecting guidelines relies on the system decomposition, interface design and embodying the design requirements, which are independent of the employed modelling language. One designer may prefer modelling with other languages such as SysML. However, the contemplative issue is the choice of modelling diagram in either of these languages. In fact, the model should present the system structure meaningfully and simply. Class diagram is one of the most utilized graphs in OO modelling by using UML. Class diagram describes a system structure with respect to decomposition hierarchy of classes and their relationships (Acheson, 2010; Bennett et al., 2005; Booch, 1994; Wagenhals et al., 2003) Thus, this research suggests modeling the logical architecture of a CES by developing the class diagram of the intended CES. SysML involves modelling with blocks instead of modelling with classes. Block diagram in SysML is very much similar to the class diagram in UML. Hence, if a designer prefers modeling with SysML, the system structure can be modeled by using block diagram. However, as mentioned earlier, this research only uses UML to describe the architecting guidelines; yet, the Meta-Architecture will be finally established by using the FSM formalism. Designing the warehouse logical architecture by application of OO method has been published recently, so, interested readers about details are referred to that work (Abdoli & Kara, 2016).

4.1.1 System Abstraction

In the domain modelling context, abstraction should reflect the abstract knowledge and abstract behavior of the domain. Abstract knowledge describes certain information, whereas abstract behavior indicates the responsibility of doing certain things in the domain (Oldfield, 2002). From OO perspective, abstraction aims to describe a system with its essential elements without focusing on their detail characteristics. Therefore, this research suggests abstracting a CES to its essential knowledge and behavior, so that, the abstraction gives an overall image about the CES domain.

Let's explain these concepts with the case study. Warehouses convert input items to outputs through warehousing processes. Warehousing processes can be divided into five main categories as given in Table 1 (Abdoli, Kara, & Kornfeld, 2016). These abstract processes can be considered as abstract knowledge as part of the structural facts in the warehousing domain.

Table 1. Warehouse abstract processes and their delivering item status

Process	Item status
Receiving	Warehouse-able items
Storing	Stored items
Picking	Picked items
Sorting	Ship-able items
Shipping	Dispatched orders

Each abstract process has its own objectives in terms of delivering a specific item status. Thus, the set of cooperating abstract processes should be able to deliver the abstract functionality of a CES. In the application of OO method, these abstract processes should be acknowledged as abstract classes. Hence, the first architecting guideline is given below;

Architecting guideline I: abstracting the CESs to certain abstract processes as such they can reflect the fact of item status transformation in the CES domain.

In a real CES, each abstract process can include several sub-processes (factual processes). Sub-processes may have different design requirements. Hence, it is not possible to derive objects from sub-process class directly. Therefore, sub-processes should be considered as abstract class too.

Generally, sub-processes need certain enablers to make the process operation possible. Reviewing widespread literatures, process enablers can be categorized into five basic categories; equipment, infrastructure, human resource (HR), material and operational policy (Gray, Karmarkar, & Seidmann, 1992; J. Gu, Goetschalckx, & McGinnis, 2007). However, a designer may come up with another categorization. The operational policy stands for the soft side of the system, such as control procedures. From OO perspective, this research suggests acknowledging the enablers as abstract classes too.

Architecting guideline II: sub-process and process enabler concept should be acknowledged as abstract elements of a CES.

The CESs operate on certain items. Thus, the item essence is part of the structural facts and can be acknowledged as an abstract class. Figure. 3 demonstrates the described abstract elements and their relationship in the warehousing domain.

Architecting guideline III: Item concept should be acknowledged as abstract element of a CES.

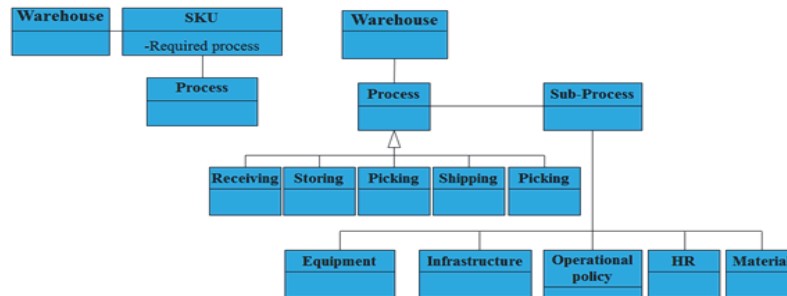


Figure 3. Warehouse abstraction to its abstract classes (Abdoli & Kara, 2016)

Thus far, the given architecting guidelines are independent of the employed modeling language. However, for the purpose of presentation, a UML dependent guideline is also given below.

Architecting guideline IV: all abstract concepts should be acknowledged as abstract classes

SysML can also describe the abstraction concept by using stereotypes. For more details, regarding modeling with SysML, the readers are referred to related literatures (OMG, 2010).

4.1.2 Use Case Development

Domain models should demonstrate the abstract usage of a system without detail information about a specific system (Booch, 1994). As described, a CES transforms items from input to the required output. Hence from item perspective, the item transformation is considered as the main usage scenario of the system of interest. This research suggests formulating this transformation based on the aforementioned abstract classes. Hence, a CES function, CES_f , is presented as a collection of certain functions for certain items, as shown in (1). Different items are formulated based on their required abstract processes, as shown in (2). Thereafter, the designer makes a global set from all sub-processes across all items. Finally, each abstract process can be formulated as a set of required sub-processes, as shown in (3).

$$CES_f = \{ \{ ITEM_j \}; j \in \{1, \dots, m\} \} \tag{1}$$

$$ITEM_j = \{ \{ \alpha_i, P_i \}; i \in \{1, \dots, 5\} \} \tag{2}$$

$$\alpha_i = \begin{cases} 1; & \text{if } P_i \text{ is needed for } ITEM_j \\ 0; & \text{otherwise} \end{cases}$$

$$P_i = \{ \{ \beta_k, SP_k \}; k \in \{1, \dots, K\} \} \tag{3}$$

$$\beta_i = \begin{cases} 1; & \text{if sub-process } k \text{ is needed for } p_i \\ 0; & \text{otherwise} \end{cases}$$

Where:

P_i : Process i, m: number of SKUs, K: number of sub-processes, SP_k : Sub-process k

Any real item can demonstrate its specific usage from the system by carrying the result of above formulation. To this end, the above formulation should be addressed as an attribute for the item class; ‘Required processes’. Therefore, a class should be derived from the item abstract class to represent each specific item. The above formulation helps to generate the vectors of {0, 1} for the aforementioned attribute. In return, each specific item

carries a vector clarifying its required processes. The presented formulation enables incorporation of use case in the emerging class diagram and not in a separate diagram. However, most of the existing works use separate diagrams (such as use case diagram) for demonstrating the system usage. This formulation also shows that how processes and items interact in the system of interest.

Architecting guideline V: Addressing the use case formulation result as the attribute of the item class

4.1.3 Specific Design

The resulted abstract classes along with their hierarchy manifest a domain model of a CES. Yet, the architecture should encompass the CES specifications in order to demonstrate the logical architecture of a specific CES. This research recommends addressing the CESs specifications in the class diagram.

As the first step, the designer can use the global set of sub-processes and derive equivalent sub-process classes from the sub-process abstract class. This research suggests utilizing the item status for clarifying the parent abstract process for real sub-processes. In fact, when an item transits to its required status from an abstract process, the subsequent activities are supporting the next abstract process. Thus, as soon as a sub-process could transform the item to the required status from an abstract process, the subsequent activities should be allocated to the following abstract process.

Each sub-process may require special types of enablers. Therefore, a subclass that is equivalent to the enabler type can be derived from the corresponding enabler abstract class. Hence, another architecting guide line can be summarized as below;

Architecting guideline VI: specific sub-processes and enablers should be addressed as derived subclasses from corresponding abstract classes

Each sub-process or each enabler may have different design requirements. For example, the required infrastructure for unloading sub-process can be a dock, which has a dock position as a design requirement. However, the infrastructure for the storing process can be a storage module, and aisles configuration can be one of its design requirements. This research suggests a novel guideline to address the specific design requirements in the model;

Architecting guideline VII: Design requirements from sub-processes or enablers can be addressed as design attributes of the equivalent classes.

In the system level, the main interactions between sub-processes are item arrival to each sub-process and its departure to the subsequent sub-process.

Architecting guideline VIII: Arrival and departure time of items should be addressed as attributes of sub-process classes

Finally, the class diagram can embody the essence of interactions in the system level by defining the aforementioned attributes too. These attributes represent an abstract behavior of the CESs. This behavior brings about item transition in the system which belongs to the structural facts. Eventually, the emergent class diagram can represent the CES logical architecture. Thus, the first task of the framework is fulfilled, as shown in Figure 1.

4.1.4 Meta-Architecture

The resultant class diagram can represent a Meta-Architecture of a CES. Indeed, the system design alternatives can be developed by deriving set of instances from the classes.

The system usage was formulated as a function of items and their required processes which both modelled as classes. As a result, any changes in the usage can be easily addressed into architecture. The change can be about introducing either new items or processes. This change can be addressed in the model by deriving new objects from the item or sub-process class. However, a system usage can also change because of changing the required processes for an existing item. In this case, the change can be addressed by modifying the 'Required processes' attribute of the existing item object. Obviously, these changes do not require changing the model structure. Nevertheless, the emergent logical architecture not only realizes the essence of holistic design, but also, the architecture is flexible regarding capturing the changes in system usage. This is the outcome of applying a proper method (OO) and its proper application. As mentioned the first gap in the existing literature was; 1-Lack of architecting guidelines. The given prescriptive architecting guidelines can successfully cover this gap.

4.2 Design Validation with Finite State Machine Based Simulation

As shown in Figure. 1, the Meta-Architecture should be established by using FSM formalism. Two issues are discussed in this section. First; how to incorporate the utilized OO semantics into FSM formalism (applying the

given guidelines in an FSM model). Second; what are the concepts that needed to be addressed in the FSM model to make sure that the design alternatives are executable.

4.2.1 Establishing System Logical Architecture in FSM Formalism

The main idea behind the aforementioned incorporation relies on seeing a CES as a machine. Basically, the domain structural facts construct the machine structure. Let's call this machine state model from now the System-State-Meta-Model (SSMM) as shown in Figure. 1.

To this end, as the first step, all abstract classes should be represented by equivalent states. As a result, the class diagram hierarchy will be satisfied by mapping the abstract classes to parent/high-level states. With regard to the case study, five states were defined at the highest level of warehouse state model; as shown in Figure. 4. These five states are equivalent to the warehouse abstract processes.

Sub-process class was defined in the next level of class diagram hierarchy. Therefore, sub-processes of each abstract process should be mapped to its equivalent sub-process-state. As shown in Figure. 4, 'Receiving' process includes 'Unloading' and 'Stacking' as its sub-process states.

Sub-process abstract class is associated to the enabler's abstract classes. Hence, each sub-process-state should include certain sub-states that are equivalent to abstract enablers. Due to space limitation, this structure is shown only for 'Receiving' and 'Storing' process of a warehouse in Figure. 4.

'Design requirements' attributes of an enabler class should be mapped into sub-states of the corresponding enabler state. Let's call these sub-states the design-requirement-states. As shown in Figure.4, 'Unloading' sub-process requires door as an infrastructure which has two design requirements; 'Door_position' and 'Door_type'. These two design requirements are addressed as sub-states (design-requirement-state) of the infrastructure state.

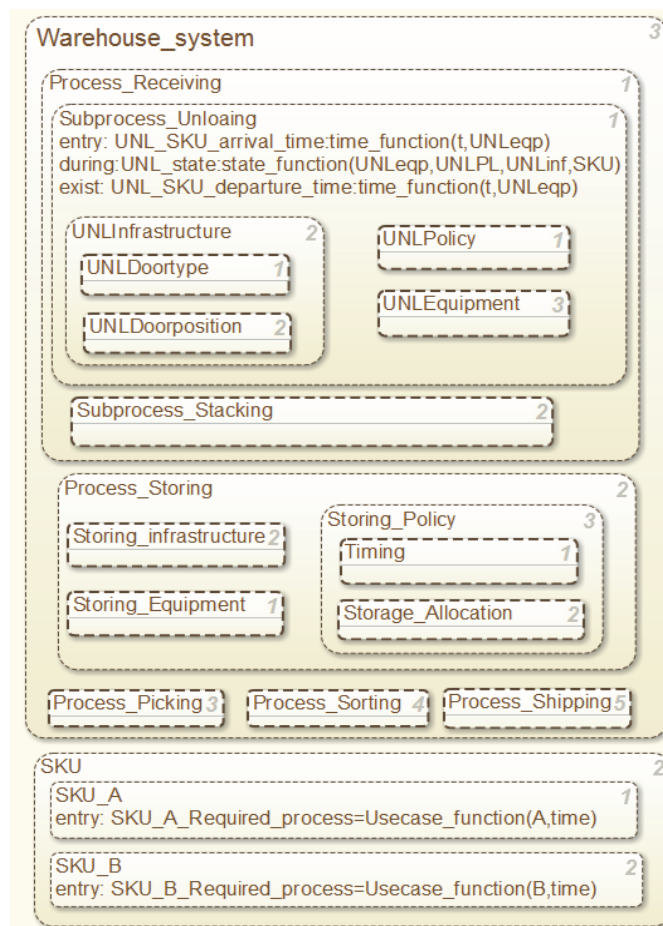


Figure 4. Warehouse state model

Arrival and departure time attributes can be mapped to certain variables of the emerging SSMM. Each sub-process should carry its own specific arrival and departure time variables. Figure.4 shows these variables for 'Unloading' sub-processes.

Item abstract class should also be mapped to a state. Accordingly, any real item can be defined as the sub-state of the item state. 'Required-process' attributes can be mapped to a variable too. Each specific item can get its real value for 'Required-process' variable by using the use case formulation. With regards to the warehousing, Stock Keeping Unit (SKU) state and its sub-states are shown in Figure. 4.

The solution space of the design requirements should be addressed as sub-states (object-state) of the corresponding design-requirement-state. As shown in Figure.5, door position is a design requirement. Accordingly, its possible options are presented as two object-states; 'Middle' and 'Corner'. Likewise, the storing process has a design requirement regarding its operational policy which is called timing. This design-requirement-state has two object-states; FIFO (First In First Out) and LIFO (Last In First Out), as shown in Figure.5. It is necessary that each design-requirement-state has one activated object-state to develop a system design alternative. In fact, a designer can activate different object-state separately and configure different design alternatives. Therefore, the designer can develop the design alternatives as instances of the emerging SSMM.

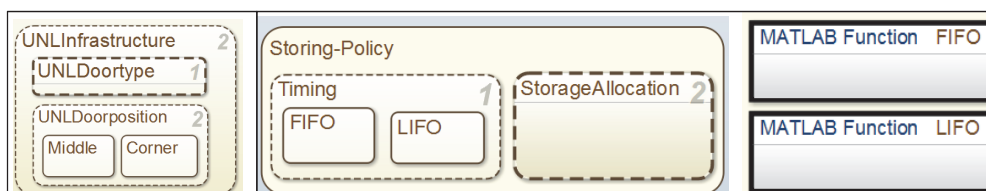


Figure 5. Examples of object-states of design requirements and state-functions

Each object-state can carry its own specific state-function. The state-function can indicate a design module for encapsulating the detail design and embodying the behavior of the object-state. For instance, FIFO-state-function and LIFO-state-function can embody their own operational logic. Accordingly, this approach enables to construct a Meta-Architecture as such the behavioral policies are not modelled as invariant. As mentioned, the third gap in existing literature was; 3-Modelling operational policies as model invariants. Thus the presented approach can overcome this gap.

Following the given guidelines, the entire class diagram can be mapped to a FSM model. The entire class diagram here refers to the classes and their attributes with respect to the class diagram hierarchy. Therefore, the resulted artefact can be officially called SSMM. This semantic incorporation of OO with FSM formalism can be summarized into following rules;

- 1- Classes should be mapped to states
- 2- Associated classes to a higher level class should be mapped as sub-states of the relevant parent state
- 3- Design requirement attributes of a class should be mapped to equivalent design-requirement-states of corresponding state
- 4- Time related attributes of process class (Arrival and departure time in processes) and 'Required-processes' attribute of item class should be mapped to variables of corresponding states
- 5- Design options for any design requirements should be mapped to object-states of the corresponding design requirement-state.

SSMM can have different building blocks such as process-state, design-requirement-states and object-state. These building blocks provide certain templates for designing similar states. The developed SSMM has a hierarchical structure. The higher level is independent of the design solution as shown in Figure.4. This level models the system structure and its design requirements. In fact, the structural facts build SSMM hierarchy by forming the nested states. The lowest level of SSMM hierarchy is solution dependent indicating the decisions for choices of the design requirements as shown in Figure.4 and Figure. 5.

4.2.2 Deriving Executable Design Alternatives

Domain behavioral laws should be addressed in the SSMM to assure alignment of the validation with domain axioms. Thus, the fashion of addressing the behavioral laws is discussed here.

In the CESs, an important behavioral law is the chronological sequence of processes operations in the system level. This law can be realized in FSM formalism by defining proper decomposition of states. If certain processes can work simultaneously, then, the relative decomposition of their abstract process should be defined as; ‘Parallel ~ AND’. The dashed lines in Figure. 4 demonstrate that the process-states can be active in the same time. If certain processes cannot not run simultaneously, then, the relative decomposition of their abstract process should be defined as; ‘Exclusive ~ OR’. Each process may need all of its enablers in the same time. Hence, the relative decomposition of processes with respect to their enablers can be defined as ‘Parallel ~AND’.

In developing an alternative, only one possible option can be allocated to each design requirement. For example, the door can be placed in the middle or corner of a warehouse wall and not both. Thus, the relative decomposition of the design requirement states with respect to their object-states should be define ‘Exclusive ~ OR’.

Sometimes, processes or enablers can operate in parallel, but, the sequence of their engagement in the operation is different. For example, although an operator and a forklift work together; the operator action triggers the forklift action. This behavioral law can be realized in the SSMM by determining the proper activation sequence for parallel states. As a result, the parallel states within one simulation time step can be activated with a different order. This feature enables examining the pre-conditions and applying the post-actions as part of the behavioral laws in SSMM. The sequential activation still guarantees calling state-functions and updating the variables of the related states at the end of simulation time step.

The CESs interact with their environments. These interactions are part of behavioral laws and should be embodied in the model. For example, a warehouse may interact with certain suppliers and customers. Hence, the concepts of supplier and warehouse can be mapped to two states in the SSMM. The warehouse may interact differently with each factual supplier or customer. Hence, each specific supplier or customer can be mapped to a single sub-state as shown in Figure. 6. The specification of interactions can be addressed in corresponding sub-states as shown in Figure. 6.

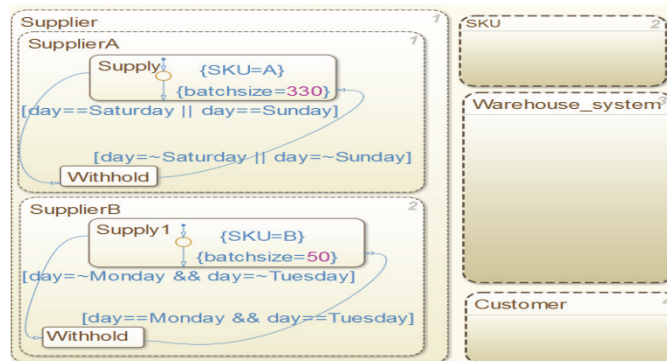


Figure 6. Example of mapping suppliers and customers to corresponding states

For example, supplier A sends a new consignment of SKU- A to the warehouse with the defined batch size. However, the ‘Unloading’ sub-process can start operating on SKUs if its required enablers are available. State-functions can control and update the availability of enablers as shown in Figure.7. Accordingly, state-functions contribute on capturing the dynamic state of the system.

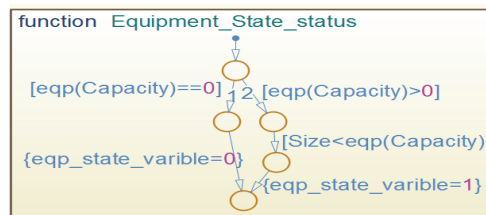


Figure 7. Example of updating equipment state status variable

Finally, the essential behavioral laws are addressed in the SSMM and its instances can be executed by addressing the specific data of a design problem. As mentioned earlier, the second identified gap in existing literature was; 2-Demonstrating the system structure and realizing the dynamic behavior in separated modelling formalisms. Thus, the elaborated SSMM can cover this gap. SSMM can be handed off to the downstream engineers. Thereupon, they can establish the concrete design in state-functions depending on the required granularity in the model. The state-functions can belong to any design discipline in form of mathematical expressions or algorithmic procedures. This realizes the essence of multi-disciplinary design in the SSMM.

5. Discussion and Conclusion

The framework utilizes OO principles to facilitate the complexity of capturing the design knowledge holistically. Accordingly, one of the main novelties of this framework is the given architecting guidelines leading to a hierarchical abstraction of the CESs. In the system level the complexity of a holistic system is broken down to the abstract processes. Defining the item status as the classifier helped to determine the belongingness of factual processes to the abstract processes in second level of hierarchy. Process enablers were defined as abstract elements of the system in the third level of the hierarchy. This abstraction can reflect domain structural facts. This hierarchy breaks the complexity and facilitates addressing all design requirements to the corresponding classes. This approach satisfies demonstrating the CES structure holistically.

Certain guidelines are given to incorporate the suggested architecting logics into FSM formalism. In fact, the CES was analogized to a state machine. Nevertheless, the aforementioned abstraction hierarchy constructs the machine structure building nested states. The design requirements were mapped to corresponding machine states and finally building SSMM. The 'design- requirement-states' and 'object-state' can stand as design libraries for system design. The emergent SSMM enables addressing the structural facts and behavioral laws of a CES. Particularly, the instances of SSMM can be DES executable. This enables observing the impact of the design decisions on the system behavior in early stages of the design. Thus, this approach facilitates early recognition of design inconsistencies. Moreover, the solution space can be observed holistically (in terms of investigating on both physical and operational aspects of the system). This approach reduces the chance of losing a good design solution. Therefore, the framework has a holistic-integrated approach.

The proposed framework formalizes the fashion of addressing the domain axioms in design artefacts without relying on any specific domain axioms. This stresses out the domain generality feature of the framework. The system use cases were embodied as model variables. This demonstrates the 'flexibility' feature of the framework with regards to changes in the system usability. The internal behavior of design options were encapsulated as state-functions of the relative design requirement. The encapsulation enables embodying the essence of multidisciplinary design in the SSMM. SSMM as a common design platform can remarkably facilitate efforts for design integration and communication.

6. Future Work

This paper focused on explaining the core of modelling approach to develop a logical architecture as such its instances can be executable. Yet, the elaboration on addressing the specification of the design knowledge in the SSMM and particularly its instances are left as the future research. Moreover, it is required to investigate regarding how to develop a systematic interconnection between state-functions so that they enable update propagation during simulation. This systematic approach should satisfy execution of all instances of SSMM. Integrating the framework with a model-based trade study is one of the main future research directions for this research. To this end, a search mechanism is required so that it can syntheses coherent design alternatives with an algorithmic approach. In return, the trade study approach can compare and prioritize the alternatives by application of proper assessment models. Ultimately, integration of such a trade study mechanism with currently proposed framework, can serve as a decision support framework in practices of CES design.

References

- Abdoli, S., & Kara, S. (2016). Designing Warehouse Logical Architecture by Applying Object Oriented Model Based System Engineering. *Procedia CIRP*, 50, 713-718. <http://dx.doi.org/10.1016/j.procir.2016.04.108>
- Abdoli, S., Kara, S., & Kornfeld, B. (2016). Application of Dynamic Value Stream Mapping in Warehousing Context. *Modern Applied Science*, 11(1), 76.
- Acheson, P. (2010, 5-8 April 2010). *Methodology for object-oriented system architecture development*. Paper presented at the Systems Conference, 2010 4th Annual IEEE.
- Albers, A., & Braun, A. (2011). A generalised framework to compass and to support complex product engineering processes. *International Journal of Product Development*, 15(1-3), 6-25.

- Alvarez, Cabrera, A. A., Foeken, M. J., Tekin, O. A., Woestenenk, K., Erden, M. S., De Schutter, B., ... 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, A. C., & Silva, S. C. (2009). *A review of design methodologies for manufacturing systems*. Paper presented at the Proceedings of the 1st International Conference on Innovations, Recent Trends and Challenges in Mechatronics, Mechanical Engineering and New High-Tech Products Development (MECAHITECH '09).
- Andersen, A. L., Brunoe, T. D., Nielsen, K., & Rösiö, C. (2017). Towards a generic design method for reconfigurable manufacturing systems: Analysis and synthesis of current design methods and evaluation of supportive tools. *Journal of Manufacturing Systems*, 42, 179-195.
- Andreasen, M. M. (1994). Modelling—the language of the designer. *Journal of Engineering Design*, 5(2), 103-115.
- Baker, P., & Canessa, M. (2009). Warehouse design: A structured approach. *European Journal of Operational Research*, 193(2), 425-436. <https://doi.org/10.1016/j.ejor.2007.11.045>
- Barbieri, G., Fantuzzi, C., & Borsari, R. (2014). A model-based design methodology for the development of mechatronic systems. *Mechatronics*, 24(7), 833-843. <https://doi.org/10.1016/j.mechatronics.2013.12.004>
- Bar-Yam, Y. (2002). General features of complex systems. *Encyclopedia of Life Support Systems (EOLSS)*, UNESCO, EOLSS Publishers, Oxford, UK.
- Bennett, S., McRobb, S., & Farmer, R. (2005). *Object-oriented systems analysis and design using UML*: McGraw Hill Higher Education.
- Bergenthal, J. (2011). Final Report Model Based Engineering (MBE) Subcommittee.
- Bienvenu, M. P., Shin, I., & Levis, A. H. (2000). C4ISR architectures: III. An object-oriented approach for architecture design. *Systems Engineering*, 3(4), 288-312. [https://doi.org/10.1002/1520-6858\(2000\)3:4<288::AID-SYS6>3.0.CO;2-F](https://doi.org/10.1002/1520-6858(2000)3:4<288::AID-SYS6>3.0.CO;2-F)
- Booch, G. (1994). *Object-oriented analysis and design with applications (2nd Ed.)*: Benjamin-Cummings Publishing Co., Inc.
- Buede, D. M., & Miller, W. D. (2016). *The engineering design of systems: models and methods*: John Wiley & Sons.
- Christophe, F., Bernard, A., & Coatanéa, É. (2010). RFBS: A model for knowledge representation of conceptual design. *CIRP Annals-Manufacturing Technology*, 59(1), 155-158.
- Cloutier, R. J., & Verma, D. (2007). Applying the concept of patterns to systems architecture. *Systems Engineering*, 10(2), 138-154.
- Cloutier, R., & Griego, R. (2008, 7-10 April 2008). *Applying Object Oriented Systems Engineering to Complex Systems*. Paper presented at the Systems Conference, 2008 2nd Annual IEEE.
- Coatanéa, E., Nonsiri, S., Roca, R., Mokammel, F., Kruck, J., & Christophe, F. (2015). Systematic search for design contradictions in systems' architecture: Toward a computer aided analysis. *Journal of Integrated Design and Process Science*, 19(1), 25-46.
- Dori, D. (2002). *Object-Process Methodology*: Springer.
- Douglass, B. P. (2016). Chapter 1 - What Is Model-Based Systems Engineering? *Agile Systems Engineering* (pp. 1-39). Boston: Morgan Kaufmann.
- ElMaraghy, H. A., Kuzgunkaya, O., & Urbanic, R. J. (2005). Manufacturing Systems Configuration Complexity. *CIRP Annals - Manufacturing Technology*, 54(1), 445-450. [https://doi.org/10.1016/S0007-8506\(07\)60141-3](https://doi.org/10.1016/S0007-8506(07)60141-3)
- Erden, M. S., Komoto, H., van Beek, T. J., D'Amelio, V., Echavarría, E., & Tomiyama, T. (2008). A review of function modeling: Approaches and applications. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 22(02), 147-169.
- Estefan, J. A. (2003). Survey of Model-Based Systems Engineering (MBSE) Methodologies (C. I. o. Technology, Trans.) (B ed., pp. 70). Pasadena, California, U.S.A.: Jet Propulsion Laboratory.
- Fishwick, P. A. (2007). *Handbook of dynamic system modeling*: CRC Press.
- 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.
- Friedenthal, S., Meilich, A., & Izumi, L. (2007). *Object-Oriented Systems Engineering Method (OOSEM) applied to Joint Force Projection (JFP), a Lockheed Martin Integrating Concept (LMIC)*. Paper presented at the Proc 17th Int Symp INCOSE.
- Gray, A. E., Karmarkar, U. S., & Seidmann, A. (1992). Design and operation of an order-consolidation warehouse: Models and application. *European Journal of Operational Research*, 58(1), 14-36. [https://doi.org/10.1016/0377-2217\(92\)90232-X](https://doi.org/10.1016/0377-2217(92)90232-X)
- Grobshtein, Y., Perelman, V., Safra, E., & Dori, D. (2007, 20-23 March 2007). *Systems Modeling Languages: OPM Versus SysML*. Paper presented at the 2007 International Conference on Systems Engineering and Modeling.
- Gu, J., Goetschalckx, M., & McGinnis, L. F. (2007). Research on warehouse operation: A comprehensive review. *European Journal of Operational Research*, 177(1), 1-21. <https://doi.org/10.1016/j.ejor.2006.02.025>
- Gu, P., Rao, H., & Tseng, M. (2001). Systematic design of manufacturing systems based on axiomatic design approach. *CIRP Annals-Manufacturing Technology*, 50(1), 299-304.
- Jepsen, A. D., Mortensen, N. H., & Hvam, L. (2014). *Architecture Descriptions. A Contribution to Modeling of Production System Architecture*: Technical University of Denmark Danmarks Tekniske Universitet, Department of Management Engineering Institut for Systemer, Produktion og Ledelse, Operations Management Operations Management.
- Komoto, H., & Tomiyama, T. (2012). A framework for computer-aided conceptual design and its application to system architecting of mechatronics products. *Computer-Aided Design*, 44(10), 931-946. <https://doi.org/10.1016/j.cad.2012.02.004>
- Koo, H.-Y. B. (2005). *A meta-language for systems architecting*. Technion, Israel Institute of Technology.
- Kossiakoff, A., Sweet, W. N., Seymour, S., & Biemer, S. M. (2011). *Systems engineering principles and practice* (Vol. 83): John Wiley & Sons.
- Mashkoor, A., & Jacquot, J. P. (2011). *Guidelines for formal domain modeling in Event-B*. Paper presented at the High-Assurance Systems Engineering (HASE), 2011 IEEE 13th International Symposium on.
- Matei, I., & Bock, C. (2012). *SysML Extension for Dynamical System Simulation Tools*: US Department of Commerce, National Institute of Standards and Technology.
- Mayerhofer, T., Langer, P., Wimmer, M., & Kappel, G. (2013). *xMOF: Executable DSMLs based on fUML*. Paper presented at the International Conference on Software Language Engineering.
- McGinnis, L. F., Huang, E., & Wu, K. (2006). *Systems engineering and design of high-tech factories*. Paper presented at the Proceedings of the 38th conference on Winter simulation.
- McGinnis, L. F., Huang, E., & Wu, K. (2006, 3-6 Dec. 2006). *Systems Engineering and Design of High-Tech Factories*. Paper presented at the Proceedings of the 2006 Winter Simulation Conference.
- McGinnis, L., Huang, E., Kwon, K. S., & Ustun, V. (2011). Ontologies and simulation: a practical approach. *Journal of Simulation*, 5(3), 190-201.
- Mijatov, S., Mayerhofer, T., Langer, P., & Kappel, G. (2015). *Testing Functional Requirements in UML Activity Diagrams*. Paper presented at the International Conference on Tests and Proofs.
- Miller, J. A., Baramidze, G. T., Sheth, A. P., & Fishwick, P. A. (2004). *Investigating ontologies for simulation modeling*. Paper presented at the Simulation Symposium, 2004. Proceedings. 37th Annual.
- Mönch, L., Lendermann, P., McGinnis, L. F., & Schirrmann, A. (2011). A survey of challenges in modelling and decision-making for discrete event logistics systems. *Computers in Industry*, 62(6), 557-567. <https://doi.org/10.1016/j.compind.2011.05.001>
- Moses, J. (2002). *The anatomy of large scale systems*. Paper presented at the ESD Internal Symposium.
- Oldfield, P. (2002). Domain modelling. *Appropriate Process Group*.
- OMG. (2010). *OMG System Modeling Language (OMG SysML) v1.2*: Object Management Group.
- Osorio, C. A., Dori, D., & Sussman, J. (2011). COIM: An object-process based method for analyzing architectures of complex, interconnected, large-scale socio-technical systems. *Systems Engineering*, 14(4), 364-382.

- 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 Computer Science*, 16, 245-254. <https://doi.org/10.1016/j.procs.2013.01.026>
- Penas, O., Plateaux, R., Patalano, S., & Hammadi, M. Multi-scale approach from mechatronic to Cyber-Physical Systems for the design of manufacturing systems. *Computers in Industry*. <https://doi.org/10.1016/j.compind.2016.12.001>
- Robinson, S. (2014). *Simulation: the practice of model development and use*: Palgrave Macmillan.
- Rouwenhorst, B., Reuter, B., Stockrahm, V., van Houtum, G. J., Mantel, R. J., & Zijm, W. H. M. (2000). Warehouse design and control: Framework and literature review. *European Journal of Operational Research*, 122(3), 515-533. [https://doi.org/10.1016/S0377-2217\(99\)00020-X](https://doi.org/10.1016/S0377-2217(99)00020-X)
- Schuh, G., Monostori, L., Csáji, B. C., & Döring, S. (2008). Complexity-based modeling of reconfigurable collaborations in production industry. *CIRP Annals-Manufacturing Technology*, 57(1), 445-450.
- SE Handbook Working Group. (2011). *SYSTEMS ENGINEERING*. San Diego: International Council on Systems Engineering (INCOSE).
- SEBoK authors. (2016). Logical Architecture (glossary)--- Guide to the Systems Engineering Body of Knowledge (SEBoK), version 1.7: Hoboken, NJ: The Trustees of the Stevens Institute of Technology ©2016.
- Simon, H. A. (1996). *The sciences of the artificial (3rd Ed.)*: MIT Press.
- Thiers, G. (2014). *A model-based systems engineering methodology to make engineering analysis of discrete-event logistics systems more cost-accessible*. Georgia Institute of Technology.
- Tomiyama, T. (2006). *A classification of design theories and methodologies*. Paper presented at the ASME 2006 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference.
- Tomiyama, T., D'Amelio, V., Urbanic, J., & ElMaraghy, W. (2007). Complexity of multi-disciplinary design. *CIRP Annals-Manufacturing Technology*, 56(1), 185-188.
- Tomiyama, T., Gu, P., Jin, Y., Lutters, D., Kind, C., & Kimura, F. (2009). Design methodologies: Industrial and educational applications. *CIRP Annals - Manufacturing Technology*, 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. *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing*, 10(04), 275-288.
- Wagenhals, L. W., Haider, S., & Levis, A. H. (2003). Synthesizing executable models of object oriented architectures. *Systems Engineering*, 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 Thechnology. (2256)
- Wang, R., & Dagli, C. H. (2008). *An executable system architecture approach to discrete events system modeling using SysML in conjunction with colored Petri Net*. Paper presented at the Systems Conference, 2008 2nd Annual IEEE.
- Wolfram, S. (1985). Complex systems theory. *Princeton: The Institute for Advanced Study*.
- Yaroker, Y., Perelman, V., & Dori, D. (2013). An OPM conceptual model-based executable simulation environment: Implementation and evaluation. *Systems Engineering*, 16(4), 381-390.
- Zheng, C., Hehenberger, P., Le Duigou, J., Bricogne, M., & Eynard, B. (2016). Multidisciplinary design methodology for mechatronic systems based on interface model. *Research in Engineering Design*, 1-24.
- Zhu, M., & Brooks, R. R. (2009). Comparison of Petri net and finite state machine discrete event control of distributed surveillance networks. *International Journal of Distributed Sensor Networks*, 5(5), 480-501.

Copyrights

Copyright for this article is retained by the author(s), with first publication rights granted to the journal.

This is an open-access article distributed under the terms and conditions of the Creative Commons Attribution license (<http://creativecommons.org/licenses/by/4.0/>).