International Journal of Business, Human and Social Sciences
ISSN: 2517-9411
Vol:9, No:5, 2015

Handling Complexity of a Complex System Design: Paradigm, Formalism and Transformations

Hycham Aboutaleb, Bruno Monsuez

***Abstract*—**Current systems complexity has reached a degree that requires addressing conception and design issues while taking into account environmental, operational, social, legal and financial aspects. Therefore, one of the main challenges is the way complex systems are specified and designed. The exponential growing effort, cost and time investment of complex systems in modeling phase emphasize the need for a paradigm, a framework and an environment to handle the system model complexity. For that, it is necessary to understand the expectations of the human user of the model and his limits. This paper presents a generic framework for designing complex systems, highlights the requirements a system model needs to fulfill to meet human user expectations, and suggests a graph-based formalism for modeling complex systems. Finally, a set of transformations are defined to handle the model complexity.

***Keywords*—**Higraph-based, formalism, system engineering paradigm, modeling requirements, graph-based transformations.

I. Introduction

USUALLY the approach we follow in a project depends on how the results will be used. To optimize the design time, it is important to have a useful framework for analyzing complex systems and study their evolution. The use of such a framework requires an understanding of the boundaries of a given system, its components, its representation, and the evolution of its model and ways of representation.

The complexity that emerges while designing and developing the system is usually the result of the multidimensionality of the system. To understand its behavior, a system is considered in the context of its environment, including interactions and interfaces. Indeed, the complexity of a system is often characterized, beyond the inherent complexity of components and their variety, by the complexity of the interaction network, from which emerges behaviors as intentional and unintentional, which may be harmful and difficult to predict and control.

With emergence of complex engineering systems, starting with defense systems after WW2, a new approach was required to handle the increasing complexity of these new complex engineering systems. The traditional reductionists approach “divide and conquer" was no longer valid to address this issue. It was necessary to define a holistic approach: every part in a system is related to every other to form a coherent whole. Such an approach analyzes better the emerging behavior as well as the interdependencies. Using a holistic approach induces considering the system as a unified whole.

This paper presents a paradigm for system engineering. The first section is dedicated to the definition of the paradigm. The second section extracts the requirements a model needs to fulfill to address the complexity issue. The third section presents formalism for a higraph-based model that will be used in this paper for system modeling. Necessary graph-based transformations that apply general concepts are also presented. A final section discusses how the proposed formalism addresses the complexity issue.

Hycham Aboutaleb is a Research Engineer in the Computer Science and System Engineering Department at ENSTA ParisTech, 828 boulevard les Maréchaux,Palaiseau, 91120 France (corresponding author e-mail: hycham.abou-taleb@ensta-paristech.fr).

Bruno Monsuez is the Director of the Computer Science and System Engineering Department at ENSTA ParisTech, 828 boulevard les Maréchaux, Palaiseau, 91120 France.

II. System Engineering Paradigm

Unprecedented levels of complexity have emerged from contemporary engineering systems. While the organic and functional aspects remain at the core of the systems engineering method, there is an urgent need to more effectively address additional aspects that are correlated to the functional spaces of the system of interest. Dimensions to be taken into account will vary from one system to another depending on the system perimeter. Therefore, it is necessary to clearly analyze the system-to-design context since a software system will not have the same environment as a network or a mechanical system. However, since a system function is performed by the system itself, various dimensions of such a system are related to each other. As different aspects become too complex for the mind to easily understand or operate with, different approaches are possible in order to better understand a complex system. Four concepts have been taken into account in this method: abstraction level, decomposition level, view, engineering perspective (Fig. 1).

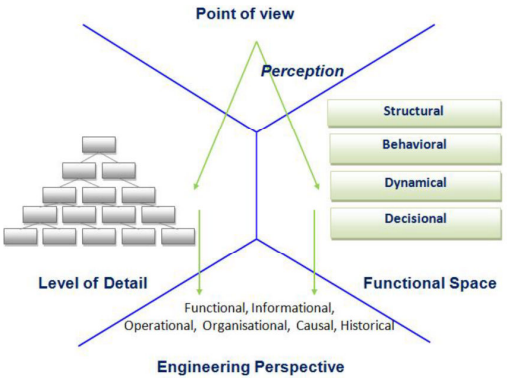

Fig. 1 System Engineering Paradigm

While abstraction level allows the observer to have a holistic view of a system but in respect to different aspects, the level of decomposition partitions the problem space and allows a localized understanding of the different dimensions of a system. As each person understands a given problem in his/her particular manner, it is of common sense that we can analyze a system from different points of view that are perfectly coherent with each other see. This approach is function-centered since it depends on the characterization of a system according to its functional spaces. [1], [2]

- *Abstraction*: view of the system that is relative to both the level of detail through decomposition and the type of information captured, but one does not need to consider these layers to understand a general phenomenon or one that is possible only in certain conditions. The functional spaces are the abstraction levels.
- *Decomposition*: isolate system components for a detailed analysis, given that all information of the context of the analyzed element is regarded.
- *Perception*: the point of view of each actor that limits or filters the available information, it allows building different models or representations of the system.
- *Engineering Perspectives*: what is needed to be taken into account to design the system. There are technical aspects (technical processes) as well as non-technical ones.

Four functional spaces are considered (Fig. 1):

- *Structural space*: characterizes the form of the system physical components and their interrelationships. Theses structural aspects include both vertical and horizontal dependencies, such as hierarchy or coupling of the component systems. Methods and tools for designing complex systems must accommodate multiplicity of scales in regard to the structural elements that comprise them.
- *Dynamical space*: characterizes changes over time, as well as time-based properties such as adaptability of the system.
- *Behavioral space*: relates to the model of the emergent behaviors resulting from the complex interconnections in order to understand how the systems will perform.
- *Decisional space*: relates to the decisions to be taken, as well as the actions expected to be performed by the system.

III. Model-Based System Engineering

Many Model-Based System Engineering (MBSE) methodologies emerged after the introduction of system engineering in the industry.

In order to better understand the key features of different methodologies, it is important to establish a terminology for better understanding these methodologies (Fig. 2):

- *Process*: sequence of tasks aiming to achieve a particular objective. Process defines what is to be done without defining how each activity has to be performed.
- *Method:* specifies *how* to perform each task.
- *Tool*: helps to accomplish of *how*. It usually supports a language that helps applying the method.
- *Methodology*: is defined "as a collection of related processes, methods, and tools". In model driven context, MBSE can be defined as a collection of process, tools and methods help to harmonize system engineering discipline.
- *Environment*: consists of external conditions, systems, or factors that have an influence on systems, actors. The purpose of environment is to put in practice the use of tools and methods of a project.

Thus, the ability to model and design a system is limited by the methods. Without a holistic approach, the cost of model construction and the effort required to integrate various system models may present critical concerns that might be reflected in the resulting system design [4].

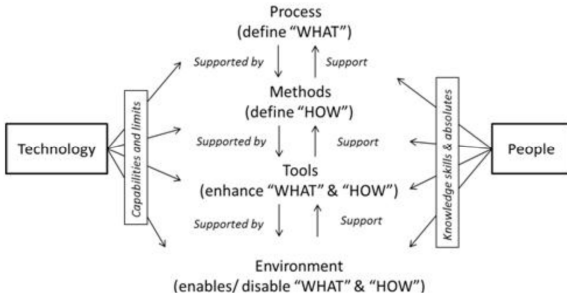

Fig. 2 System Engineering Framework [3]

IV. System Representation and Modeling

Model-based development has been adopted more or less in development of complex systems today. To understand this trend, it is necessary to focus on the properties of complex systems to design and to the needs of the stakeholders involved in the development of these complex systems. A model has a clear purpose: to help designing the system of interest. Modelers must exclude all factors not relevant to the problem to ensure the project scope is feasible and the results timely. The value of the modeling process begins early on, in the problem definition phase. The modeling process helps focus diagnosis on the system of interest.

A. Modeling Issues

When developing complex systems, two main problems arise:

- The need to address all the aspects of the system of interest (to design and develop). [5]
- The need to share the knowledge between people involved in the process. [6]

To match these needs, model-based system engineering is necessary. However, the need of a model-based approach induces new issues:

- Trustworthiness of the model: how close the model is to the reality?
- Understandability of the model: is the model perceived and understood the same way by people?
- Usefulness of the model: does the model help to get the desired results?

1. Trustworthiness of System Model

Given the limited cognitive capabilities of humans, we use models of the properties of the system and its context/environment that are of relevance and interest, and disregard details considered irrelevant for the system design and development. A model is thus a deliberate simplification of reality with the objective of explaining a set of selected properties of the real system that is relevant for the purpose of its development. This model starts first with a mental process to capture relevant information, then the information captured is expressed through means to be communicated. This information is the minimum information necessary to have a satisfactory understanding of the perceived real system and environment. [7]

2. Understandability of System Model

The understandability of a model depends on how an individual perceives the model that he/she is going to use. Two people share the same *mental model* if they have similar descriptions, explanations, and predictions of the system of interest. Specifically, models allow people to similarly predict and explain the behavior of the system of interest, to recognize and remember relationships among its components and with its environment, and to construct expectations for what is likely to occur next.

3. Usefulness of System Model

To help ensure the utility of shared mental models, a distinction is often drawn among different types of mental models, normally based on their underlying content. In order to be useful, a model shall facilitate accomplishing a task and allow each individual to work effectively as a member of the team [8]. Thus, a model would be considered effective if team performance is increased. According to [9], a team performance is related to the *taskwork mental model similarity*, the *teamwork mental model similarity*, the *taskwork mental model perceived accuracy*, and the *teamwork mental model perceived accuracy*.

Moreover, it shall allow engineers to reuse and share past solutions. This has an additional advantage: inexperienced engineers benefit from the work of more experienced ones and are able to work at their quality levels.

B. Model Complexity and Hierarchy

Since systems are inherently complex, to obtain a model that is trustworthy, understandable, and useful, it is necessary to architecture the complexity. As it is described in [10], there is a form of organized complexity in systems. To handle large amounts of data, it is often useful to have a classification or an order. One effective way to classify a set of elements is to use a hierarchical organization of this set of elements, introducing sometimes new order relations among the elements. With the hierarchy, in addition to be able to handle elements together, it becomes possible to handle subsets of elements together. There are two ways how to organize hierarchically a set: grouping and encapsulation.

- Grouping: It is possible to group items based on similar properties or characteristics.
- Encapsulation: It is possible to encapsulate many elements within a single element of a higher level and then consider only the properties of this element when an analysis is performed.

Therefore, to handle complexity of the real system, its model should be the result of a simplification strategy consisting in:

- Conceptual chunking: refers to the formation of a higher-level concept that captures the essence of the problem-at-hand and reduces the complexity by omitting irrelevant detail and reducing its dimensionality [11].
- Segmentation: refers to the decomposition of a complex system into smaller parts that can be studied in isolation, in order that the capacity limitations of the human mind are avoided.

Consequently, we can identify two types of models hierarchies. On one hand, there is the *generalization*, i.e. hierarchy of types. The word type refers generally to a representation that gathers main properties of objects that have common characteristics [12]. One type allows to group elements with common characteristics. The mechanism of sub-typing induces a hierarchy: an entity type T2, derived from type T1 has at least all the properties of an entity type T1.

On the other hand, there is *aggregation*. The word aggregation refers generally to a representation that gathers elements into another higher-level element to hide them when necessary. The higher-level element that encapsulates its contained elements has properties that are the emerging properties at this level due to the contained elements. Other names like nested hierarchy or container hierarchy are also common. Encapsulation decreases the complexity of the system model [13].

Finally, the hierarchy has an additional advantage: depending on the selected level, it is possible to observe different points of view.

C. Modeling Requirements

As presented in the previous section, the modeling approach must be powerful enough to express all relevant properties of a system. In order to enable the modeling of systems with the characteristics mentioned in the previous section, the modeling approach needs to meet the following requirements [14]-[17]:

- Analogy: A model shall be analogous to existing models or at least to a *conceptual* model to avoid conceptual "clash". If there is such analogy, it should be pointed out to help establishing links to the existing conceptual model of a user and facilitates reasoning: human reasoning is less based on an application of formal laws of logic than on memory retrieval and analogy.
- Utility: A model shall serve a useful well-defined purpose.

- Stability: A model shall be usable uniformly in many different contexts without any qualification or modification.
- Projection: A model shall support diverse and integrated views on the system under development. By this way, different aspects of the system can be independently analyzed and specified.
- Modularity: A model shall have a hierarchical organization of its composing elements. Elements might be aggregated and encapsulated in a higher-level element to facilitate analysis and eventual reuse.
- Compositionality: A model shall allow deducing the properties of a system from the properties of its subsystems. It is essential for the reuse of existing components, and enables to incrementally build systems out of modularly specified parts.
- Abstraction: A model shall enable capturing properties of a system that are needed to understand one of its aspects without paying attention to the details or the other aspects of the system.
- Refinement: A model shall start with high-granular descriptions and allow to incrementally refining them into more detailed ones. A refined model shall guarantee all the properties of the abstract model.

V. Higraph-Based Model

Graphs have been naturally used to represent and model problems since the emergence of computer science. Graph-based models give a visual and intuitive representation, as well as with required accuracy. They are a well-suited means to describe in a natural way all kind of systems, where nodes describe system entities and edges describe relations between them [18]. However when it comes to representing complex systems, the absence of hierarchy is certainly one of the main defaults of graph-based representations. [19]

A. Definition

A higraph is a graph extended to include notions of depth and orthogonality and was introduced by Harel in [20], [21]. In other words:

Higraph = Graph + Depth + Orthogonality

Definition (Higraph). A higraph is a quadruble $H = (B; E; \rho; \Pi)$ where:

- B is the set of blobs (or nodes);
- E is the set of edges.
- ρ is the hierarchy function. It assigns to each blob $b \in B$ its set of sub-blobs $\rho\ (b)$.
- Π is the orthogonality (or partitioning function) defined as $\Pi : B \to 2^{B \times B}$, associating with each blob $b \in B$ some equivalence relation $\Pi(b)$ on the set of sub-blobs, $\rho\ (b)$.

By its definition, the depth, shown by a higraph is defined by the enclosure of one node within another. Thus, it is possible to develop a higraph from a tree (Fig. 3).

B. Transformations

To meet the modeling requirements defined previously, we need to define a set of higraph-based transformations that will fulfill these expectations.

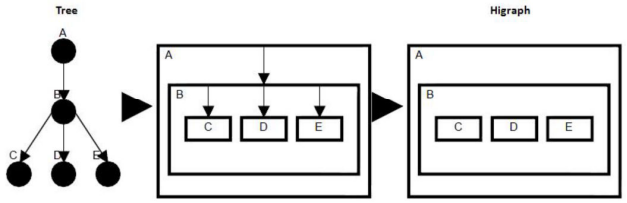

Fig. 3 Developing a higraph from a tree

1. Generalization

First, we need to define the Type Higraph M_Π associated to the Higraph M. To achieve this it is necessary to define first *generalization*.

Definition (Generalization).

Let M_Π be a Type Higraph.

Let M be a Model Higraph.

Let $g : M \to M_\Pi$ a morphism that associates to each element (object, flow, attribute) x of the Model Higraph M to its type, with M_Π, the Model Type Higraph.

We have:

- $\forall x \in M, g(x) \in M_\Pi$;
- $\forall x \in M, g(\rho(x)) \subset \rho(g(x))$;
- $\forall t \in M_\Pi, g(\Pi_t(x) \subset \rho(t)$.

Besides, $M_\Pi = (B; E; \rho; \Pi)$ is a higraph where:

- $E = 0$;
- $\forall x \in B, \Pi(x) = \rho(x)$.

2. Aggregation

Now we define the function *aggregation*. This allows a higher-level element to encapsulate its contained elements while having its properties that are the emerging properties at this level due to the contained elements.

Definition (Aggregation)

Let M be a Model higraph.

Let x be a model node.

Let y_i be model nodes such that $y \in \rho(x)$.

The aggregation function f_{agg} maps a set of elements y_i to a single element x:

$f_{agg} : M \to M$ such that $f_{agg}(y_1, ..., y_{|\rho(x)|}) = x$.

This function is used to represent an object as a black box, i.e. without its children elements. Its corresponding inverse function is the *decomposition* function.

3. Decomposition

Definition (Decomposition)

Let M be a Model higraph.

Let x be a model node.

Let y_i be model nodes such that $y \in \rho(x)$.

The decomposition function f_{dec} maps a single element x to a set of elements y_i:

$$f_{dec} : M \rightarrow M \text{ such that } f_{dec}(x) = \{y_1,...,y_{|\rho(x)|}\}.$$

This function is used to represent an object as a glass box, i.e. with its children elements. Its corresponding inverse function is the *aggregation* function.

4. Refinement

Definition (Refinement)

Let M be a Model higraph.

Let x, y be two model nodes. An element x is said to be refined by an element y if x contains y, i.e.:

- $y \in \rho(x)$

This transformation allows refining an element in the model.

5. Filtering

The Model Views are obtained by filtering the Type Higraph. Thus, there exists a filtering function that can be applied: *Filtering*. The Model Views are obtained by filtering the Type Higraph.

Definition (Filtering)

A filter function is a function $f : M_{\Pi} \rightarrow V$, where M_{Π} is the Type Higraph and V is a Model View, which either preserves nodes in the Type Higraph or removes them.

Thus:

- $V \subset M_{\Pi}$
- $\rho(V) \subset \rho(M_{\Pi})$

This transformation allows extracting a view containing elements of the model -that are of the same type- and their decomposition.

VI. Conclusion

System complexity is usually due to the recursive intricacy and the interactions between the subsystems. However, human behavior makes a system far more complex and complicated due to the perception: stakeholders usually do not have the holistic view that enables understanding of the system and taking into account all the factors and elements that can are related to the design process.

To handle the complexity of a complex system design, we proposed a better understanding by defining a system engineering paradigm. We identified the several levels a system can have, and define the functional spaces a system usually has. It is expected that this approach could be used for any system.

In this paper, we also defined the requirements a system model needs to meet to be trustworthy, useful and understandable. One of the most important issues addresses was the model complexity. To handle the complexity, it is necessary to architecture the model. Hierarchy is the most intuitive way to address this issue. Two main types of hierarchy have been defined in that purpose. The proposed higraph-based formalism and its associated transformations allow representing complex systems while meeting the modeling requirements defined beforehand.

It is now relevant to ask a new question: how to evaluate the complexity of a system model. This would be especially useful to measure the perceived system complexity (i.e. the system model complexity). It would also prove the efficiency of such an approach.

Another question would be the implementation of this approach. Literature review and industrial experience indicate that system engineers are still in need of a modeling language that is simple and intuitive to support many tasks in system engineering and architectural reasoning.

References

[1] Hycham Aboutaleb, Samuel Boutin, and Bruno Monsuez. Handling scenarios complexity in model-based design. *Concurrent Engineering: Research and Applications, Special issue: Complex systems design and Management*, 20(2):1–20, 2012.

[2] Sara Sadvandi, Hycham Aboutaleb, Cosmin Dumitrescu. "Negotiation Process from a Systems Perspective". *In Proceedings of CSDM 2011*, Paris, France, 2011. Springer Verlag.

[3] Martin, James N., *Systems Engineering Guidebook: A Process for Developing Systems and Products*, CRC Press, Inc.: Boca Raton, FL, 1996.

[4] Hsueh-Yung Benjamin Koo. *A Meta-language for Systems Architecting*. PhD thesis, MIT, USA, (2005)

[5] Håvard D. Jørgensen. *Interactive Process Models*. PhD thesis, Norwegian University of Science and Technology Trondheim, Norway, 2004.

[6] Mark Sean Avnet. *Socio-Cognitive Analysis of Engineering Systems Design: Shared Knowledge, Process, and Product.* PhD thesis, MIT, USA, 2009.

[7] Ali Mostashari. *Stakeholder-Assisted Modeling and Policy Design Process for Engineering Systems.* PhD thesis, MIT, USA, (2005)

[8] J.A. Cannon-Bowers, E. Salas, and S.A. Converse. *Shared mental models in team decision making.* In Individual and Group Decision Making, pages 221–246. N.J. Castellan Jr, (1993)

[9] Kevin Forsberg and Harold Mooz. *The relationship of systems engineering to the project cycle.* Journal of Applied Psychology, 85(2):273–283, 2000.

[10] Herbert A. Simon. "The architecture of complexity." *Proceedings of the American Philosophical Society*, 106(6):467–482, 1962.

[11] Graeme S. Halford, Rosemary Baker, Julie E. McCredden, and John D.Bain. "How many variables can humans process?" *Psychological Science*, 16(1):70–76, 2005.

[12] Luca Cardelli and Peter Wegner. "On understanding types, data abstraction,and polymorphism." *ACM Comput. Surv.*, 17(4):471–522, 1985.

[13] Valerie Ahl and T. F. H. Allen. *Hierarchy Theory - A Vision, Vocabulary, and Epistemology*. Columbia University Press, 1996.

[14] G. S. Halford, J. Wiles, M. S. Humphreys, and W. H. Wilson. "Parallel distributed processing approaches to creative reasoning: Tensor models of memory and analogy." *In Proceedings of the AAAI Spring Symposium*, Palo Alto, California, USA, 1993. T. Dartnall, S. Kim, and F. Sudweeks.

[15] Hermann Kopetz. The complexity challenge in embedded system design. In *ISORC*, pages 3–12, 2008.

[16] Marc Bouissou. *Gestion de la complexité dans les études quantitatives de sûreté de fonctionnement de systèmes*. Lavoisier, 2008.

[17] Alexander Harhurin, Judith Hartmann, and Daniel Ratiu. *Motivation and formal foundations of a comprehensive modeling theory for embedded systems*. Technical report, Technical University of Munich, 2009.

[18] Leen Lambers. *Certifying Rule-Based Models using Graph Transformation*. PhD thesis, Technical University of Berlin, Germany, 2009.

[19] Frank Drewes, Berthold Hoffman, and Detlef Plump. "Hierarchical graph transformation". *Journal of Computer and System Sciences*, 64(2):449–283, 2002.

[20] David Harel. Statecharts: "A visual formalism for complex systems." *Science of Computer Programming,* 8(5):231–274, 1987.

[21] David Harel. "On visual formalisms." *Communications of the ACM*, 31(5):514–530, 1988.