A Higraph-Based Formalism for System Modelling Language ArKItect

Hycham Aboutaleb and Bruno Monsuez

***Abstract*—This paper highlights the requirements a model needs to fulfill to match human user expectations. We introduce higraph-based formalism for system modeling. Due to the hierarchical and compositional nature of higraghs, representations based on higraphs can capture all the relevant aspects of the system to model. A system modeling language, arKItect is also introduced. We show how this language is based on the proposed formalism.**

***Index Terms*—Higraph-based formalism, system modeling, system engineering framework, tool.**

I. Introduction

Since the introduction of System Engineering in industry model-based development has been adopted more or less in development of complex systems. A model has a clear purpose: to help designing the system of interest. However, when developing complex systems two main problems arise: 1) The need to address all the aspects of the system of interest (to design and develop) [1]. 2) The need to share the knowledge between people involved in the process [2]. To match these needs, model-based system engineering is necessary. However, the need of a model-based approach induces new issues:

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

Besides, with the emergence of model-based system engineering methodologies, a framework has been proposed for better understanding. As illustrated in Fig. 1, a terminology is defined:

- *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.

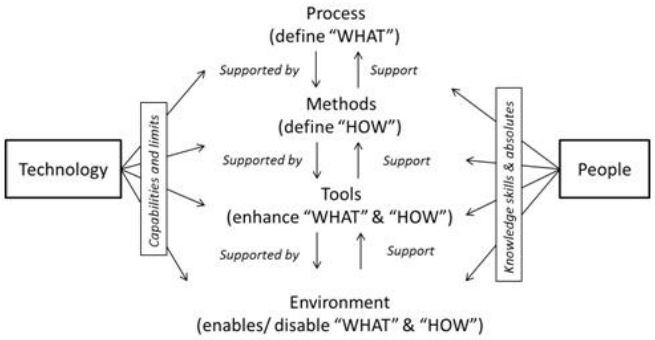

Fig. 1. System engineering framework [3].

Therefore, the system modeling language and the associated tool is extremely critical to address issues raised above. This paper presents a new system modeling tool and proposes a corresponding higraph-based formalism. It finally discusses in what extent the proposed solution addresses the mentioned issues.

This paper is organized as follows: Section II gives the mathematical definition of higraph model and introduces the underlying semantics; Section III presents the tool arKItectand proposes a corresponding higraph-based formalism; Section IV discusses if itcan be used in complex systems development.

II. Higraph Semantics

A. The Hierarchy Issue

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 well-suited means to describe in a natural way all kind of systems, where nodes describe system entities and edges describe relations between them [4]. However, when it comes to representing complex systems, the absence of hierarchy [5] is certainly one of the main issuesin graph-based representations.

Since systems are inherently complex, to obtain a model

Manuscript received January 1, 2014; revised March 3, 2014.

The authors are with Computer and System Department, ENSTA ParisTech, 828 Boulevard des Maréchaux, 91120 Palaiseau, France (e-mail: hycham.abou-taleb@ensta-paristech.fr)

that is trustworthy, *understandable*, and *useful*, it is necessary to architecture the complexity. As it is described in [6], 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.

Consequently, we can indentify 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 [7]. One type allows to group elements with common characteristics. The mechanism of sub-typing induces a hierarchy: an entity type T_2, derived from type T_1 has at least all the properties of an entity type T_1.

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 [8].

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

B. Higraph

A higraph is a graph extended to include notions of depth and orthogonality and was introduced by Harel in [9], [10]. 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.

III. ARKITECT

A. Overview

ArKItect is a software language for the representation and design of systems, designed to assist users in the implementation of the key processes of Systems Engineering. It offers a support to the activities of requirement management, functional analysis and system decomposition, risk analysis, cost management, project management.It is a graphical multi-scale tool for defining Diagram Domains Specific Languages (DSL) and using them in different applicative domains. ArKItect Diagram Domain Specific Language is made up of three layers: D_2, D_1 and D_0 layer (Fig. 2) [11]

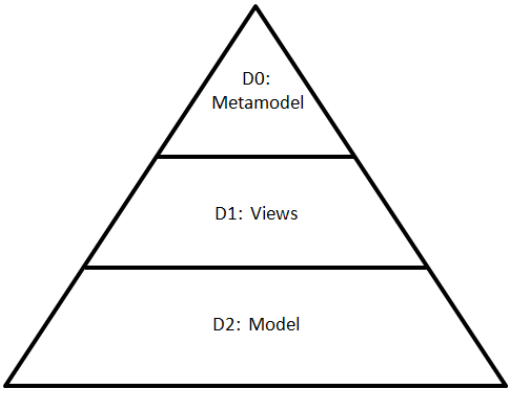

Fig. 2. ArKItectmodelling pyramid.

- At the D_2 level, the Type Diagram describes the properties of the objects types that are used in the model. It also describes the relationship between these objects types (hierarchy).
- At the D_1 level, a Tree View Diagram describes the model elements to be displayed with their layout properties. This description respects the relationships defined by the Type Diagram.
- At the D_0 level, we have diagrams expected by end-users in their modeler.

Given the application domain and the size of the project, the environment needed for development and the way people work can vary a lot. Therefore, arKItect is a framework to develop system development tools and the three levels, D_2, D_1 and D_0 are supported by the framework together with a user administration tool allowing granting a variety of access rights to the different users. ArKItect is founded on an evolution of the object model supporting a different approach to flows representation and handling. It is a relational object model because flows between objects are first class citizens. The theoretical added value of this object model is that all the DSL are built upon very few features, in comparison to UML or other derived languages. Of course, the more diagrams are added; the more difficult it is to master a DSL.

B. Mathematical Foundations

1) Basic Elements

In arKitect, only three elements exist: Object, Flow, Attribute.

Object: The basic entity in arKItect is called an object. Each object can also have one or several named data fields, called attributes. An object is unequivocally defined by its name and type. Thus, there can be two objects of different types with the same name while it is not possible to have two different objects that share the same name and type. However, it is possible to have several representations of the same object: a single object can thus appear at different nodes in the model, in different diagrams, or in different treeviews. However, the object in question remains always the same. In particular, its name and properties remain the same, and if the treeviews use the same filter, the visible children are identical (Fig. 3). While some of different object representations stem from the fact that the project can be filtered in different manners, some of them are created by multiple object references. In arKItect, it is possible to create several references to the same object. From a mathematical point of view, an object is a higraph; it can contain another object, a flow or an attribute. It may also be shared by others, i.e. it may have several parents. An object is defined by its type, its name, and has graphical properties (shape, color, borderline).

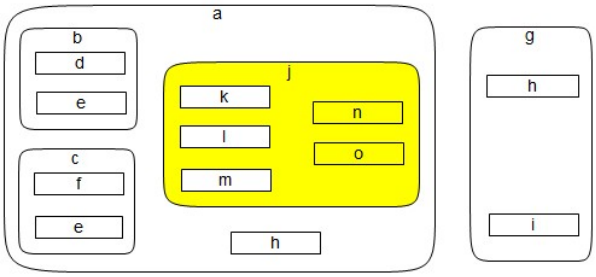

Fig. 3. Example of objects in arKItect

Flow: In arKItect, flows are considered as objects. Flows also have some special graphical properties not applicable to standard objects. They are composed of 2 higraphs and an edge between them. These two higraphs are similar in everything (contain the same elements and have the same decomposition), except the fact that one is the tail and the other is the head. As a flow is an object, it has all the same characteristics: it can have children and every action available to an object is also available to a flow. Thus, a flow may contain an object, another flow or an attribute. The mathematical definition of a flow is:

Definition (Flow).

A flow is an element composed of 2 higraphs G and H, and an edge between them such that:

- G is the tail.
- H is the head.
- $\rho(G) = \rho(H)$.
- $\Pi(G) = \Pi(H)$.

To connect two objects in arKItect, one of them shall contain the flow producing object (the tail) and the other one shall contain the flow consuming object (the head). Therefore, a flow is defined by the location of its producing and consumming objects.

- Attribute: In arKItect, objects can have additional semantic properties called attributes. An attribute is a higraph with no depth; thus it can contain no other element. The mathematical definition of an attribute is:

Definition (Attribute).

An attribute is a higraph G such that: $\rho(G) = 0.$

2) D_2: Type Matrix

D_2 level represents the Type Matrix: it is the description model of modeling language. This level contains the types of elements necessary to use the modeling language, as well as the hierarchical relations between them.

The metamodel in arKItect, also called Type Matrix is a higraph (or a hierarchy of higraphs) where there is no edge (Fig. 4). An element in the Type Matrix is a higraph and it can be shared between other higraphs (i.e. other elements of the Type Matrix). These elements can be objects, flows or attributes. All elements in these higraphs have the same type from a higraph point of view, i.e. they are all considered of the type "node".

Definition (Type Matrix).

A type matrix $T = (B; E; \rho; \Pi)$ is a higraph where:

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

An element in the Type Matrix is said to be *refined* if it contains other elements. Thus, at the Type Matrix level, only one transformation can be applied: *Type Refinement*.

Definition (Type Refinement).

Let T_1 and T_2 be two type higraphs.

A type T_1 is said to be refined by a type T_2 if T_1 contains T_2, i.e. $T_2 \in \rho(T_1)$.

On Figure 4 on one hand"Type_1" is refined by "Type_2" and "Type_1" (itself),and on the other hand "Type_2" is refined by "Type_1" and "Type_2" (itself).

3) D1: Tree Views

D_1 level represents the Tree Views: it defines the set of views that are used for system modeling. This level contains the set of diagrams expected to be used by engineers. The Tree View in arKItect is a subset of the Type Matrix, i.e. it is a higraph (or a hierarchy of higraphs) where there is no edge and this higraph is included in the Type Matrix higraph (as illustrated in Fig. 5).

Definition (Tree View).

Let T be a higraph representing the Type Matrix.

$V_i$is a Tree View such that:

- V_i is a higraph;
- $V_i \subset T$

Usually we have: $T = \bigcup V_i$

The Tree Views are obtained by filtering the Type Matrix. Thus, there exists a filtering function. At the Tree Views level,

one transformation can be applied: *Filtering*. The Tree Views are obtained by filtering the Type Matrix.

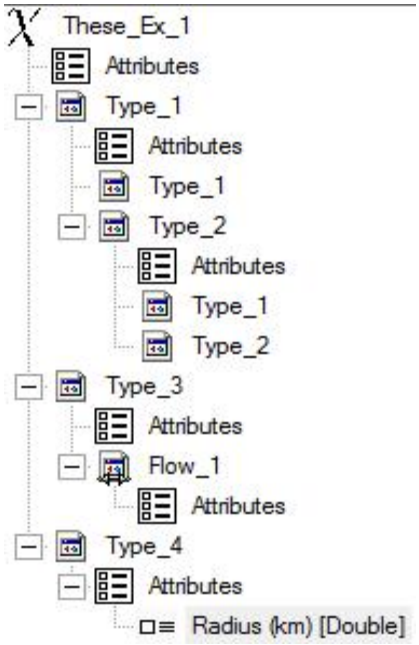

Fig.4. Example of ArKItect Type Matrix.

Definition (Filtering)

A filter function is a function $f : T \to V$, where T is the Type Matrix and V is a Tree View, which either preserves nodes in the higraph or removes them.

Thus:

- $V \subset T$
- $\rho(V) \subset \rho(T)$

4) D0: Model

D0 level represents the Model: it is the set of diagrams that represent the system. This level contains the set of diagrams used by engineers. The Model in arKItect is a rooted higraph, i.e. a higraph with a unique top-level element, called root, that contains objects, flows and attributes. An example illustrated in Fig. 6 where objects of different types have different colors. A model shall respect the type matrix. Thus, all elements have a decomposition that shall match their corresponding type decomposition. At the Model level, the following transformations can be applied:

a) Generalization

Definition (Generalization)

Let M_{Π} be a Type Matrix **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:

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

b) Model refinement

Definition (Model Refinement)

Let M be a Model higraph. Let M_{Π} be the associated Type Matrix; Let $g : M \to M_{\Pi}$ the morphism that associates to each element (object, flow, attribute) of the Model higraph to its type. 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.:

- $y \in \rho(x)$
- $g(y) \in g(\rho(x))$

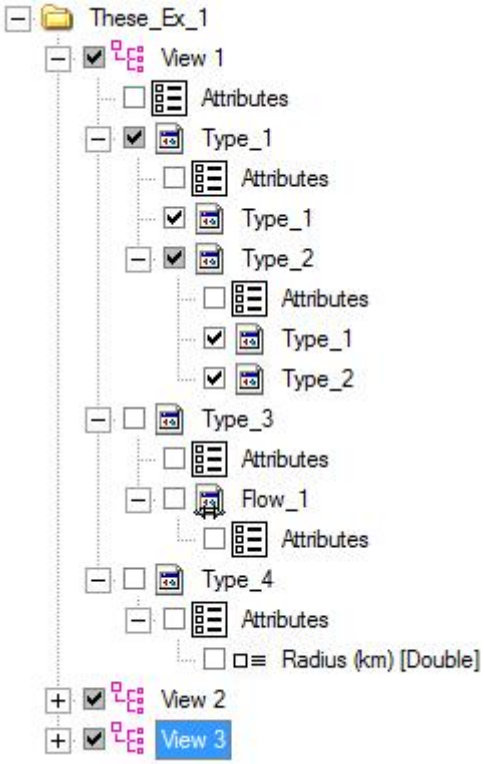

Fig. 5. Example of ArKItecttree view.

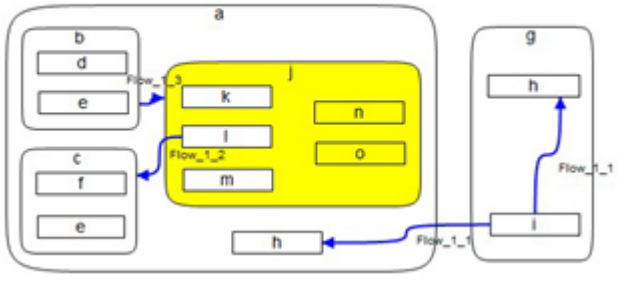

Fig. 6. Example of a model.

c) Aggregation

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.

d) 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 aggregation 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.

IV. Discussion

The well-organized single typed hierarchy of diagrams permits holistic understanding of the system and its environment effortlessly, while concurrently showing several aspects of the system. The model is easily navigable thanks to the single hierarchy of diagrams, supporting fully hierarchical organization of information.

Besides, the model permits to organize hierarchically information and to have the black box/glass box view according to the user need.

Although arKItect is fundamentally a sound tool for modeling due to its graphical (in a mathematical sense) nature and flexibility, there is one significant drawback: the complex process of arranging the nodes and edges in a visual layout that maximizes communication of information to an end-user. The problem can be difficult, even in a small sized model, and becomes critical in a large scale (industrial) system. In a large system of many components, behaviors, and domains, there most likely exists an incredible amount of overlap among the nodes and edges that make up a model of such a system.

Finally, as a language, arKItect is intended to model general-purpose complex systems. Thus it is very flexible. On another hand, this flexibility shall be controlled: defining a metamodel is not an easy task and usually requires very experienced specialists.

References

[1] H. D. Jørgensen, "Interactive process models," PhD thesis, Norwegian University of Science and Technology Trondheim, Norway, 2004.

[2] M. S. Avnet,"Socio-cognitive analysis of engineering systems design: shared knowledge, process, and product," PhD thesis, MIT, USA, 2009.

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

[4] L. Lambers, "Certifying rule-based models using graph transformation*,"* PhD thesis, Technical University of Berlin, Germany, 2009.

[5] F. Drewes, B. Hoffman, and D. Plump, "Hierarchical graph transformation*," Journal of Computer and System Sciences*, vol. 64, no. 2, pp. 449–283,2002.

[6] H. A. Simon, "The architecture of complexity," in *Proc. the American Philosophical Society*, vol. 106, no. 6, pp. 467–482, 1962.

[7] L. Cardelli and P. Wegner, "On understanding types, data abstraction, and polymorphism," *ACM Comput. Surv*, vol. 17, no. 4, pp. 471–522, 1985.

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

[9] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of Computer Programming*, vol. 8, no. 5, pp. 231–274, 1987.

[10] D. Harel, "On visual formalisms," *Communications of the ACM*, vol. 31, no. 5, pp. 514–530, 1988.

[11] B. Langlois, D. Exertier, and G. Devda, "Toward families of qvtdsl and tool," in *Proc. 6th OOPSLA Workshop on Domain-Specific Modeling DSM'06*, pp. 87–97, 2006.

Hycham Aboutaleb is a PhD student in the electronics and computer engineering Department at ENSTA Paristech, France. He has a MSc in System Engineering from Ecole Polytechnique, France and a BSc in electronics and communications engineering from Cairo University, Egypt. He worked as a system engineer at Knowledge Inside where he participated in the implementation of system engineering processes with several companies and in the development of system engineering software tools. His current research areas include design methodologies and tools, system engineering, safety and dependability engineering, optimization and complexity reduction, and quality control.

Bruno Monsuez graduated in 1989 from Ecole Polytechnique. He received his PhD in computer science from the Ecole Polytechnique in 1994. He is now the director of the electronics and computer engineering Department at ENSTA-Paristech. His current research interests are focused on developing and enhancing hierarchical compositional mathematical models that can be used to represent hardware and software components of complex embedded systems as well as formal verification techniques that allow a cojoint verification of functional and non-functional properties of the software as well as of the hardware on which the software is expected to run.