Using HiGraph to define a Formal Integrated System Modeling Framework that ensures Complete System Consistency

Anis Otmane Cherif[1], Bruno Monsuez[1], Vladimir-Alexandru Paun[1,2]
[1]*U2IS - ENSTA ParisTech,*[2]*Five Rescue Laboratory*
828 Boulevard des Marchaux, F-91120 Palaiseau
anis.otmane-cherif@ens-paris-saclay.fr,
bruno.monsuez@ensta-paristech.fr,
vladimir-alexandru.paun@ensta-paristech.fr

Michel Nakhlé
CS SI - Activité Aéronautique, Énergie & Industrie.
22, avenue Galilée
92350 Le Plessis Robinson
michel.nakhle@cs-si.fr

***Abstract*—The evolution of the design of complex systems leads to increasing complexity and requires the joint analysis and refinement of different views of the same system which generally consist of: (1) A functional view that describes the main features of the system; (2) An implementation view that allocates functions on system constituents; (3) A non-functional view ensuring that properties such as quality of services, real-time constraints... are satisfied by the system; (4) As well as a dysfunctional view that defines the reliability requirements. Despite the complexity of systems, the consistency of views when exploring the solution space must be ensured. For example: (1) A decision on the required availability may induce new functions or involve redundancy of function/constituent; (2) Another difficulty comes from the fact that the functions are being described using different formalisms, therefore the system engineer must always be able to handle all the following aspects: the availability or reliability models that are mainly based on probabilistic models, the functional view that can be expressed using finite state machines or by event models; the quality of the services that can be expressed either by using a probabilistic approach or an approach based on a bounded set... The work described in this paper focuses on the implementation of a unified industrial modeling process using the graphical language of Hi-Graphs, a specific class of hyper graphs, in support to SysML. This process brings in addition functional views, taking into account, at all stages of the life cycle, non-functional and dysfunctional views of the system in order to make the right choices / compromises in terms of both software engineering and formal verification. It provides end-to-end assurance that the system meets the requirements and contracts associated with service quality during the process of exploring and refining the solution among the different views of the system. It also offers multiple semantics so that existing modeling languages and tools are taken into account.**

***Keywords*-Formal System Modeling; Hypergraphs; Multiple-Views Modeling; Solution Exploration; Complex System; Systems Engineering.**

I. Introduction

The evolution of complex system design induces an increasing complexity of functionalities that are performed by the system. This increasing complication impacts the complexity of the functional, the logical as well as the technical architectures. Additional requirements like safety, dependability, but also reusability or maintainability increase the number of constraints that should be taken into account when exploring the design solution space. As a consequence, novel methodologies as well as new design approaches emerge with the goal not only to ensure that the system works according to the omnibet expectations but to ensure that the resulting design is optimal, exhibits a safe behavior and is reliable.

System modeling plays a key role in Systems Engineering. System modeling as well as Model Transformations allow a comprehensive and coherent representation of all the aspects of the systems going from the requirements to the logical structure, the functional behavior, the resource consumption, the safety and reliability assignments... It helps to improve the exploration of the solutions, the evaluation of the design during the exploration phase. It also helps during the implementation and integration phases to ensure that the implemented system conforms to the system model. System modeling also greatly helps to communicate between and to coordinate the activities of the stakeholders, reducing the risk of exploring inadequate solutions and the risk of implementing inadequate or non-conforming solutions. With a complete system model as well as sound model transformations that simplify the exploration of the system solution space, the implementation should be straightforward. Current Modeling approach concentrates on modeling different paradigms. Depending on the type of paradigms to model, the system engineer will provide a specialized model expressed in a given modeling language that is well suited for the addressed paradigm. Each model can be seen as an independent model and the system designer has to ensure that the different models are consistent. There are very little to no meta-models or tools that help to ensure the consistency of the views.

In this paper, we introduce HiGraphs, a specific class of Hypergraphs in roder to provide a meta-modeling language that can encapsulate current modeling languages (*eg*, state-

chart, grafcet, Petri Nets, *etc*) and that is capable of representing and handling jointly different paradigms; Typically the functional model, the non-functional model, the dysfunctional model and the implementation model of different kinds of architectures. HiGraphs provide a formal model that natively supports multi-formalism, typically this allows using different ways of modeling and integrating them in the model as sub models; HiGraphs also ensure the consistency of the multiple views, allowing the system designer to relate and characterize the impact of the requirements on the impacted physical and logical components. This allows to ensure that at the end of the refinement process the solution is complete and fulfills the whole requirements. This joint modeling approach allows to aggregate the qualitative study, which eases checking requirements enforcement for each step of the integrative verification.

The multi-formalism aspect of HiGraph also refers to the ability of HiGraph based model to be transformed to others existing known formalism by model transformations, according to two techniques: graph transformations and hybrid transformation approaches :

- Based on graph grammars [17], graph transformations are techniques and formalisms directly applicable to model transformation. In this approach the source and target models are represented as graphs. This visual notation also makes it possible to express the transformation rules in graphical form. It is formal and well founded on mathematical bases (like the theory of graphs and formal grammars), allowing to verify certain properties of the transformation. A graph grammar thus allows for a formalism to model a transformation [18], AToM3, AGG et GReAT [19]
- Hybrid approaches are a combination of different techniques. In particular, we can find approaches using both declarative and imperative rules. ATL (ATLAS Transformation Language) is an example of this approach [20].

If we consider a concurrent distributed system, the HiGraph based model may be transformed to a Petri-Net representation so that a model checker can formally check that for instance no deadlock can occur.

The paper is structured as follows: after a brief introduction of current system modeling languages (Section 2), we formally introduce the HiGraph (Section 3) and their key properties (Hierarchy and Orthogonality). We then briefly introduce the working example (Section 4) and the modeling process on the working example (Section 5). We finally discuss the interest of the approach and the integration of the existing formalism (Section 5). Finally, we show future works and conclude the paper.

II. State of art

The literature provides a wide variety of systems modeling languages (e.g., SysML, UML, arKItect, Statechart, Petri Net, ...etc.). Key properties of modeling languages are simplicity, visual and semantic flexibility, domain and user specificity, semantic preciseness, customization, compositionality, multiple views, integrability, extensibility, textual property, versioning, completeness check property. SysML and UML suffer from a lack of formal semantics (e.g., confusing and/or absence of a hierarchy definition among allocations), difficult navigation as they do not support fully hierarchical organization of information (e.g., get lost in the different system views), connections between components, functions and requirement lack of consistency (e.g. nothing ensure that a requirement is satisfied or not), UML focuses too much on (software) system design and need better requirements modeling. SysML partly addresses those aspects but still maintains different fully separated views based on different formalisms (Class Diagrams, Sequence Diagrams, ...). arKItect's propose a powerful extensible graphical language that may expose different views. However the inherent complexity of arranging the edges and nodes as defined in arKItect into a visual layout that can be manipulated by an end-use reduces the number of supported transformations and can be seen as one of the most severe limitation of arKItect. Petri Nets are suitable for modeling the system's behavior. Petri Nets like HiGraphs are highly extensible, with the ability of adding many attributes like colors, time, probabilities. However Petri Net are not hierarchized, the more complex a system becomes, the larger is the number of states and transitions; this leads to a state explosion that either the designers or the formal verification or simulation tools have difficulty to cope with.

III. HiGraph-Based Model

Graphs have been used to represent and model problems since the emergence of automated systems. Graph-based models give a visual and intuitive representation of the different concepts or components as well as the interactions and relationships between all the components or concepts that compose the system. It can represent different paradigms with the required accuracy. Therefore, Graphs are often the most natural way to describe a great variety of systems since we can associate to the graph representation different semantics that assign to nodes and edges completely different meanings depending on the type of the paradigm we want to model.

HiGraphs can be seen as an extension of ordinary graphs. Vertices are decomposed according to different orthogonal planes. Typically, the basic decomposition includes the AND and the OR planes. However the number of orthogonal planes is not limited. This extension is achieved by defining (1) a function that associates the depth or hierarchy to sub-graphs (depicted by encapsulation) and (2) an orthogonality function that maps the graph to is projection on each orthogonal plane, the graph is seen as the cartesian product of the sub-graphs (depicted by

juxtaposed partitions that are separated by dashed lines, each partition representing the projection to a plane). In the spirit of [7], [8] and formalization work of D-HiGraph [16], we may write:

$$HiGraph = Graph + Hierarchy + Orthogonality \quad (1)$$

A. Mathematical definitions

Definition 1 (HiGraph) A HiGraph is a *quadruple* $H = (B, E, \rho, \pi)$ defined by [9]):

- a finite set $B = \{b_1, b_2, \ldots, b_n\}$, whose elements are called blobs, where each blob can represent according to the application a class, an attribute, an element of a system, a function, etc.;
- a finite set $E = \{e_1, e_2, \ldots, e_m\}$, whose elements are called edges (arcs), an arc when it is defined can connect any blob to another to represent different relationships (e.g. physical connections, logical connections, functional connections, etc.). For an edge $e = (b_1, b_2)$, b_1 denotes the source blob and b_2 the target blob by edge e;
- ρ is the hierarchy function;
- π is the partitioning (or orthogonality) function.

B. Hierarchy & Orthogonality

1) Formal presentation:

Definition 2 (Hierarchy function & Hierarchy) Let $\rho : B \rightarrow 2^B$ be the hierarchy function. ρ maps each blob $b \in B$ to the set of sub-blobs $\rho(b)$ that compose the blob b (i.e., it defines the direct descendants of a blob b). Literally a blob in another defines hierarchy and shows aggregation and composition relationships.

Definition 3 (Partitioning function) Let $\pi : B \rightarrow 2^{B \times B}$ be the partitioning (orthogonality) function, so that it defines for each blob $b \in B$ an equivalence relation $\pi(b)$ on the set $\rho(b)$ of b's descendants. This induce equivalence classes, denoted $\pi_1(b), \ldots, \pi_{k_b}(b)$ and specifies the breakup of b into its orthogonal components. Let ρ^* and ρ^+ be the reflexive transitive closure, and the irreflexive transitive closure, respectively of ρ. It is required that blobs in different orthogonal components of b to be disjoint and for any pair of blobs x and y that satisfy $x \notin \rho^*(y)$, $y \in \rho^*(x)$ and $\rho^*(x) \cap \rho^*(y) \neq \emptyset$, the blobs in $\rho^*(x) \cap \rho^*(y)$ be contained in exactly one orthogonal component of x and one orthogonal component of y.

Definition 4 (Orthogonality) Two blobs x and y are orthogonal, denoted $orth(x, y)$, if $x = y$ or if there exist blobs b, c and d such that b contains r orthogonal components and there exist $1 \leq i, j \leq r$ with $i \neq j$, such that $c \in \pi_i(b)$, $x \in \rho^*(c)$, $d \in \pi_i(b)$ and $y \in \rho^*(d)$. A set of blobs X is orthogonal if any two elements therein are orthogonal.

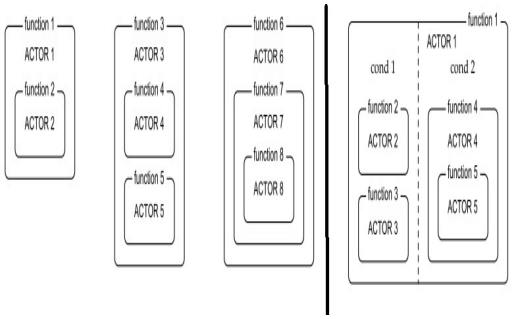

Figure 1. Example of Hierarchy and Orthogonality

2) Use of hierarchy and orthogonality notions:

Hierarchy. Referring to hierarchy definition blobs can be contained in other blobs. The outer blob that contains the included blob is called the ancestor; the included blob is called the descendant. For example, in Fig.1 (left part) function 2 is included in function 1 so function 1 is the ancestor of function 2 (super-blob) which is the descendant of function 1 (sub-blob). A blob with no descendants is called an atomic blob. Sub-blob that are included into a blob implies that to the function, the property, the activity or the concept as defined by the sub-blob is required by the super-blob. The sub-blob is an essential element of the super-blob. No sub-blob that does not contribute to the definition as exposed by the super-blob should be included in the super-blob.

Orthogonality. A blob can be decomposed into orthogonal blobs (the blob is represented by all the orthogonal partitions that are separated by a dashed line). Each orthogonal partition is composed of at least one blob. This blob defines the projection of the main function, activity, concept exposed by the blob that is relevant to the specific partition. The function π defines the semantics meaning of the relationship. Orthogonality in HiGraphs can represent (but is not limited to) separation, logical/structural partitioning, synchronized behavior, concurrent behavior, etc. In fig.1 (right part), for instance, the outer function (function 1) has been partitioned into two orthogonal components using a dashed line. This HiGraph means that ACTOR 1 performs function 1 but it needs either function 2 or function 3 and function 4. Moreover, function 5 is included in function 4 so ACTOR 4 needs the function performed by ACTOR 5 to carry out function 4. Each partition of a blob can be active or not, to show this behavior, each partition can have a condition. Another kind of orthogonality will be presented in the next sections, for example, requirements will be organized into groups that can be assigned to teams having expertise in an area relevant to those requirements.

3) HiGraph based transformations: To apply the paper approach, we define the following transformations on HiGraphs.

The first transformation is defined as a top-down trans-

formation. For that, we introduce the notion of type. A type denotes that objects having the same type T share a set of common characteristics that are defined by the type T. Grouping elements by types allows to group elements that share the same common characteristics that is denoted by the type. Refining the entities implies deriving type since for each type we refine, we have to conform to the characteristics associated to the entity and we add additional properties, constraints to the refined entity. If we suppose that an entity A$'$ has type T_2; the refined entity A$'$ has type T_2 and that T_2 derives from T_1, this implies that the entry A$'$ fulfills all the common characteristics that are defined by the type T_1 and may fulfill additional characteristics as defined by the type T_2.

Definition 5 (Decomposition) Let H be a HiGraph. Let x be a blob. Let y_i be the blobs such that $y_i \in \rho(x)$.
The decomposition function[1] f_{dec} maps a single element x to a set of elements y_i. $f_{dec} : H \rightarrow H$ such that $f_{dec}(x) = y_1, \ldots, y_{|\rho(x)|}$.

Definition 6 (Type HiGraph) Let the *quadruple* $T_\pi = (B_{T_\pi}, E_{T_\pi}, \rho, \pi)$ be a Type HiGraph such that:

- $E_{T_\pi} = \emptyset$ no edge are defined;
- $\forall x \in B_{T_\pi}, \pi(x) = \rho(x)$ (i.e., all the elements have the same type).

Definition 7 (Type Refinement) Let T_1 and T_2 are Type HiGraphs. A type T_1 is refined by type T_2 if T_1 contains T_2, i.e., $T_2 = \rho(T_1)$. This transformation allows also refining any non atomic element in the model.

The second transformation is aggregation. This is the dual transformation to refinement. Aggregation gathers the elements properties into a higher-level aggregating element (the aggregated element) and hides the lower-level elements.

Definition 8 (Aggregation) Let H be a HiGraph. Let x be a blob. Let y_i be blobs such that $y_i \in \rho(x)$.
The aggregation function f_{agg} maps a set of elements y_i to a single element x[2]. $f_{agg} : H \rightarrow H$ such that $f_{agg}(y_1, \ldots, y_{|\rho(x)|}) = x$.

Views are defined by filtering the global model (the HiGraph of type HiGraphs) accord to the type of the elements. This allows to map the model to a common set of interesting views like functional, non-functional, dysfunctional or architectural views.

Definition 9 (View) Let H_T the HiGraph of Type HiGraphs. V_i is said a view if:

- V_i is a HiGraph
- $V_i \subset H_T$

In general $H_T = \cup V_i$.

Definition 10 (Filter function) A filter function is a function $f : H_T \rightarrow H_T$, where H_T is the HiGraph of Type HiGraphs such as:

- $f(H_T) \subset H_T$
- $\rho(f(H_T)) \subset \rho(H_T)$

[1]This function makes the content of the object visible (white box), i.e. all the children elements are visible [10]

[2]This function hides the content of the object (blackbox), i.e. all the children elements are hidden. This function is the inverse function of *decomposition* function [10].

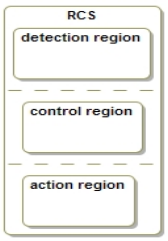

Figure 2. The RCS functional view HiGraph

IV. Working Example

As an example, we consider a Controlled Railway Track Cross Section (RCS) throughout this paper to illustrate how to use HiGraphs to model and refine a typical system.

The RCS describes a decentralized controller that autonomously activates visual and auditory signals and closes a railroad barrier at a railroad crossing when a train is approaching the railway track cross section. This controller autonomously opens the railroad barrier and stops the visual and auditory signals when the train has left the railway track cross section. This system is typical of a broad family of digital control applications. A set of senors detect the trains that are approaching. When the trains get detected by the sensors, the controller process the information and either generates a command to close the railroad barriers or to maintain the railroad barriers closed if the railroad barriers are already closed. This ensures that the railway track gets isolated from the road and that no car will be present on the railway track cross section.

A. Characteristics of Models

A system model offers a representation of the different aspects of the system. It should be capable of providing a representation of the conceptual as well as the physical properties of a system. He should be capable to offer a simple and adequate representation that matches the specifier or designer thoughts. The modeling process should allow to identify the elements that are relevant for modeling; the modeling process should also offer the tools to define the relations between those elements. Modeling elements as well as relationship between elements should be done by using well-defined and adequate languages. For instance, at very high abstraction level, the requirements are written in natural language. When refining the design, the requirements get

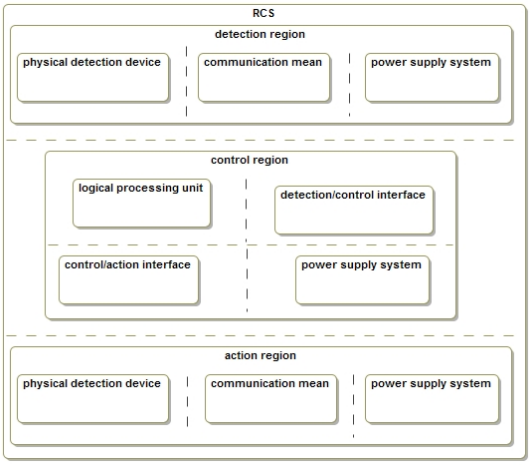

Figure 3. The RCS implementation view HiGraph

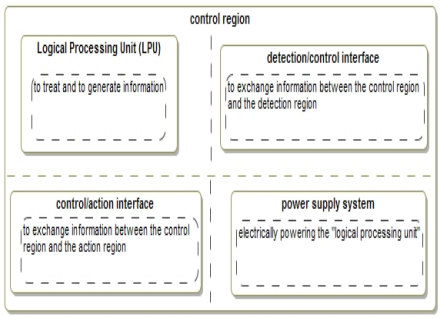

Figure 4. Mapping basic functions to control region HiGraph

replaced by components that are expressed in semi-formal or formal languages, whose semantics are mathematically rigorous and support behavioral simulation or execution.

The presented Hi-Graph based model has been conceived (1) to support multiple level of abstractions, (2) to offer the full expressive power required to model all the relevant issues (3) to offer to the specifier or designer a way to specify or express his requirements and system decomposition based on their current practices, (4) to offer a comprehensive representation of the multiple views based on a graphical representation, (5) to offer a rigorous mathematical structure to support model transformation, (6) to offer a rigorous mathematical semantics (see [9]) that is required to simulate system behavior and perform analysis of the system.

B. Modeling Views

When building a model, the designer converts non-formalized ideas into concrete and tangible descriptions [13], [14]. In the presented approach, different views describe the system specification according to the nature of the properties. Typically, the designer will introduce at least three views: the functional, the dysfunctional as well as the physical one corresponding to the implementation. In the current example, we introduce the functional, the dysfunctional, the physical as well as the non-functional (temporal) view, since we need to ensure that the delay between detecting the train, activating the signals, closing the barrier, stopping the signals and releasing the barrier are adequate.

1) Functional view: The functional description of a system specifies the system's capabilities. It details the functional components or activities, that the system is capable of carrying out. It also describes the interaction among the components or activities as well as with the environment. The environment may be modeled in this view or the interface to the environment may be specified and the environment get modeled or simulated into another view. The functional view defines the information that are exchanged between the components or activities. It also defines the logical interconnection of the components or activities. The intent of the functional view is to help to decompose complex functions into less complex functions that can be distributed to different processing components or activities that will implement the sub-function or execute the sub-task. When refining the functional view, the different components or activities that described the functions carried out by the system are decomposed into sub-components or sub-activities. This refinement process will be carried out till there are no benefit to decompose the sub-function or the sub-task. Components or activities can be organized into a hierarchy and can also be grouped by orthogonality. High-level basic functions or activities will be specified using structured natural language. When refining the components or activities, the sub-components or activities will be specified either by more precise definition in the same formalism or using a more formal mechanism like a formal description language or programming code.

2) Implementation view: The implementation captures the system architecture. The initial view of the implementation view captures a typical control command view at a high abstraction level. During the exploration of the solution space, the components get refined by more precise implementation components. This refinement process ends with the final components that correspond to the final components that support the functions that get implemented into the system. The components can be hardware components, logical components (software running on hardware) or even human activities. The implementation view also specifies how the components are interconnected and interact. These components may eventually materialize as hardware, software, or even human tasks. The relationships between the implementation view and the functional view ensure that all the functions get mapped to the components and that all the required

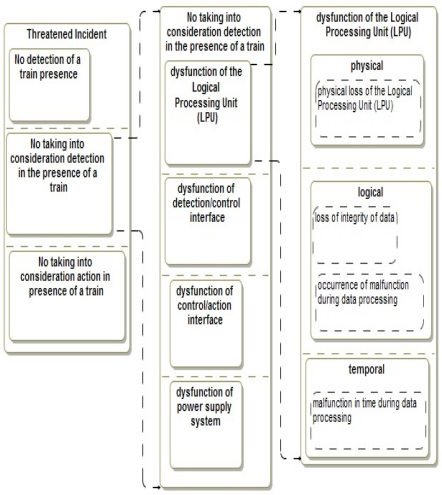

Figure 5. Non-functional/Dysfonctional view HiGraph

functions will be implemented. The relationships between the implementation view and the non-functional view (resp. dysfunctional view) ensure that all the temporal behaviors or the components (resp. faults affecting the components) are taken into account. As in the other views, they are organized into a hierarchy and grouped by orthogonality.

To illustrate functional and implementation views function based decomposition, we consider the controlled RCS example. We start by describing the main functionality, we organize them through groups using orthogonality function (dashed lines), we obtain three regions: a detection region, a control region, and an action region (see fig. 2). Now we apply the decomposition transformation that implements components to each one of these regions, we obtain the HiGraph shown in fig. 3, for example the detection relies on physical device like sensor, communication mean to transmit data signal and power supply to feed this sub-system. Lets take the control region and map basic activities for each region's components textually in dashed box (fig. 4): the logical processing unit (LPU) receives the given information, processes it and generates the resulting commands that may be a controlled signal; detection/control interface (control/action interface) allows exchanging information between respectively the detection region and the control region (the control region and the action region); finally the power supply feeding this sub-system. This operation could be repeated for each element in other regions.

3) Non-functional view: The non-functional description describes non-functional properties that can be quantified. When modeling non-functional views, we concentrate on runtime non-functional properties. For example, we will not

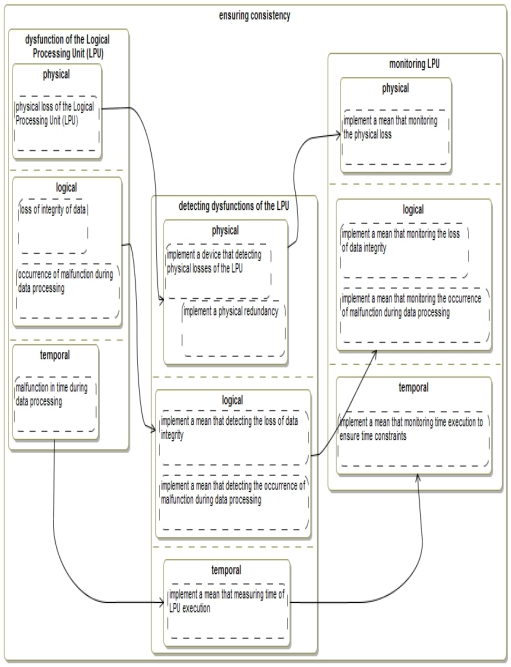

Figure 6. Ensuring the consistency of the multiple views HiGraph

model in this view re-usability, testability and modifiability.[3]. Typical non-functional properties are the performance of an action, the delay of a computation. It can also be the number of transactions that can be served at a given time, the power consumption,... Non-functional views may mix properties coming from different aspects of the attributes. When specifying temporal properties, we specify constraints of the time when an event is generated as well as the delay of an operation. Typically non-functional view should support decomposition of the typical expressed non-functional property according to the different aspects of the expressed non-functional attribute. A real-time system may express a constraint on the loop time; this constraint will decompose into a constraint on computation performance as well as a constraint on communication latency.

4) Dysfunctional view: The dysfunctional view of a system specifies the system's failure occurrences. It details the dysfunctional events, that lead the system to failure, and how these failure modes propagate through the system. The main method for describing the dysfunction of a system in our approach is that of decomposition, by which the system is viewed as a collection of hazardous events, organized into a hierarchy and grouped by orthogonality. Each event may be decomposed into sub-events, repeatedly, until the system has been specified in terms of basic events. Those basic events are the basic events that determine the origin

[3]However, additional views supporting those attributes may be introduced to support those attributes

of the critical failure.

Between the non-functional and dysfunctional view, it remains a close link, to illustrate the different views, we propose to study a critical scenario in terms of safety based on the RCS example. The scenario consists of the non-sealing level crossing in the presence of a train, in other words, the occurrence of non-closure of the RCS at the passage of a train. By applying the decomposition to this hazardous non-functional event. We obtain a HiGraph (fig. 5 first graph from left-hand) composed of three sub-events: the non-detection in the presence of a train, the non-taking into account of the detection in the presence of a train, and the non taking into account of the action in the presence of a train. The occurrence of at least one of these events is sufficient for the occurrence of the higher-order event, the non-closure of the RCS. These non-functional sub-events are grouped by orthogonality into three distinct regions, which recall the decomposition and partitioning of the functional view (fig. 2). Taking the non-functional sub-event: not taking into consideration detection in presence of a train by decomposition we obtain four events that can induce this higher-order event (fig. 5 second graph from left-hand): dysfunction of the LPU, dysfunction of detection/control interface, dysfunction of control/action interface and dysfunction of power supply systems. These sub-events are grouped by orthogonality, in this case structural. Now let us explore the event dysfunction of the LPU, the decomposition leads to sub-events that may be dysfunctional (e.g., physical loss of LPU, loss of integrity of data) or non-functional (e.g. occurrence of malfunction during data processing, malfunction in time during data processing), partitioning suggests a grouping according to the type of the events: physical, logical and temporal (see fig. 5 first graph from the right). This methodology allowed a top-down modeling until reaching the basic event that leads to the dreaded top events as well as specifies its type (in our e.g., physical, logical and temporal).

C. Ensuring the consistency of the multiple views

In the previous sections we discussed different kinds of views, and their representations. A full-fledged model of a system may consist of many views. It is interesting to note that the number of views is not fixed and that there is not a limited set of predefined views. In the working example, we have introduce four views, however a system designer can introduce as many views as he feels adequate. All the views must not be specified when starting the exploration of the solution space, a view may be added when reaching a certain exploration level. Another benefit of using the HiGraph-based modeling is the ability to support different formalisms when modeling the system, since there is no single formalism that can handle all the different aspects to be modeled. For instance, dysfunctional view is mainly based on probabilistic models, functional view may be expressed using finite state machines or event-based models, non-functional view may be expressed either using probabilistic approach or bounded set approach. HiGraphs allows putting things together, typically a HiGraph that contains different HiGraphs each one representing a typical view that has many elements inside, edges and orthogonality that can respectively show connections between the multiple views and ensure the consistency of the multiple views during the exploration of the solution space for instance a decision on the expected availability may either induce new functions or implies component duplication.

We now recollect the full model of our RCS example (see fig. 6). To achieve this, we combine the various views that are described in the previous sections. The non-functional/dysfunctional view of the RCS that appears in fig. 5 5 (first graph from left-hand in fig. 6), the functional view of detecting non-function/dysfunction following the safety scenario (second from left-hand in fig. 6), and the implementation view of monitoring (first graph from right-hand in fig. 6). The different perspectives are depicted by the same HiGraph. Lets consider the event not taking into account the detection of the train form the dysfunctional/non-functional view, this suggest a dysfunction/non function of the control system, the decision to guarantee the safety implies new functions to the functional view; for instance robustification/redundancy aspects to the physical level, the ability to detect faults (non-functional and/or dysfunctional) for logical aspects, and the ability to detect dormant faults (latency) for temporal aspects. In order to be able to check/compare the new induced functions we need the implementation view of monitoring aspects whether physical, logical and temporal.

V. Discussion

A. Interest of the approach in terms of following requirements

The interest of the approach is mainly in providing a framework that handles multiple views through orthogonality, ensure the consistency while refining abstraction levels and exploring solution space, for instance it brings adaptable implementation model according to the requirements (e.g., in Fig.2 sensor must detect a train - induce checking the failure of sensor, this leads to the use of monitoring architecture), the new needed requirements and/or the safety while designing. This also allows to check that each requirement is respected during the conception.

B. Integrating existing formalisms

The multi-formalism aspect of HiGraph refers to the ability of HiGraph-based model to be transformed to other

existing known formalism by model transformations, according to two techniques: graph transformations and hybrid transformation approaches presented in the introduction section. If we consider a concurrent distributed system, the HiGraph based model may be transformed into a Petri-Net representation so that a model checker can formally check that for instance no deadlock can occur.

VI. Future works

A future direction worth pursuing is providing dysfunctional modeling by a complementary view, making relation between the availability and the safety; also introduce the notion of time by adding time information to the model. Another future direction worth pursuing is potentially adding the notion of cybersecurity (e.g., authentication of sensors), integrating existing formalisms and provide tools that handle HiGraphs.

VII. Conclusion

We formally introduced the HiGraphs and their key properties (Hierarchy and orthogonality). We then briefly introduced the working example and the modeling process on the working example. We finally discuss the interest of the approach and the integration of the existing formalism.

Acknowledgment

Acknowledgements: This research carried out in the Project DEPARTS is partially supported by grants from BPI France within the French government R&D program PIA/FSN.

References

[1] H. Aboutaleb, *Applying HiGraph-Based Model to System Engineering - Methodology, Formalism and Metrics*, PhD in Computer Science, Paris, École Polytechnique, 2015.

[2] P. Letelier, *A Framework for Requirements Traceability in UML-based Projects In Proc. of 1st International Workshop on Traceability in Engineering Forms of Software Engineering.* In conjunction with the 17th IEEE

[3] S. Bernardi, S. Donatelli, and J. Merseguer, *From UML sequence diagrams and statecharts to analysable Petri net models.* In WOSP 02 : Proceedings of the 3rd international workshop on Software and performance, pages 3545, New York, NY, USA, 2002. ACM.

[4] T. Holvoet and P. Verbaeten, *Petri charts : an alternative technique for hierarchical net construction.*, in Proceedings of IEEE Conference on System, Man, and Cybernetics, 1995.

[5] Z. Hu and S. M. Shatz, *Mapping UML diagrams to a Petri net notation for system simulation*, in SEKE, pages 213219, 2004.

[6] Sh. Yao and S. M. Shatz, *Consistency checking of UML dynamic models based on Petri net techniques*, in CIC, pages 289297, 2006.

[7] D. Harel, *On visual formalisms* in Communications of the ACM, 31(5):514530, 1988.

[8] D. Harel, *Statecharts: A visual formalism for complex systems* in Science of Computer Programming, 8(5):231274, 1987.

[9] O. Grossman and D. Harel (1997). *On the Algorithmics of HiGraphs*.

[10] H. Aboutaleb and B. Monsuez. *Handling Complexity of a Model in System Design: Framework, Formalism and Metrics*, in Procedia Manufacturing. 3:1981-1988. 2015.

[11] M. Naklé and B. Monsuez *RTCE-SAFECOMP: Introduction a la Methodologie SAFECOMP et Rapport d'étape*, Version 1.3c du 11/9/2018.

[12] J. A. Stankovic and K. Ramamritham, *What is predictability for real-time systems* in Real-Time Systems, 2:247254, 1990.

[13] D. Drusinsky and D. Harel *Using Statecharts for Hardware Description and Synthesis*, in IEEE Trans. Computer Aided Design of Integrated Circuits and Systems 8 (1989), 798-807.

[14] K. Fogarty, *System Modeling and Traceability Applications of the HiGraph Formalism*.MS thesis. Institute for Systems Research. University of Maryland, MD 20742. May 2006.

[15] M. Raitner, *Efficient visual navigation of hierarchically structured graphs*, 2006.

[16] J. L. de la Mata and M. Rodriguez, *D-HiGraphs Functional Modeling*, R-2010-009. May 4, 2010. Autonomous Systems Laboratory. UPM-ETS Ingenieros Industriales. Jos Gutirrez Abascal, 2.28006 Madrid, SPAIN.

[17] A. Gerber, M. Lawley, K. Raymond, J. Steel and A. Wood, *Transformation: The Missing Link of MDA*, In : Graph Transformation. Volume 2505 of Lecture Notes in Computer Science, Springer- Verlag (2002) 90-105 Proc. 1st International Conference. Graph Transformation, Barcelona, Spain 2002.

[18] H. Vangheluwe, J. de Lara and P. J. Mosterman, *An Introduction to Multi-Paradigm Modelling and Simulation*, Tutorial, In Proceedings AI Simulation and Planning AIS-2002. Pages 9-20. Lisbone. Avril 2002.

[19] G. Taentzer, K. Ehrig, E. Guerra, J. de Lara et al. *Model Transformation by Graph Transformation : A Comparative Study*, in Model Transformations in Practice Workshop at MoDELS, Montego, 2005.

[20] F. Jouault, F. Allilaire, J. Bzivin, I. Kurtev and P. Valduriez, *ATL: a QVT like transformation language*, in Companion to the 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006, October 22-26, 2006, Portland, Oregon, USA