ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **27**, Number 1, 2024, 65–80

Software Platform Based on the hLARM Formalism for Modeling Complex Systems

Dragos Constantin POPESCU* and Ioan DUMITRACHE

Faculty of Automatic Control and Computers, University Politehnica of Bucharest, Splaiul Independentei No. 313, 060042 Bucharest, Romania

Email: dragos.popescu@upb.ro*, ioan.dumitrache@upb.ro

* Corresponding author

Abstract. The problem of defining, analyzing and reasoning behaviors, facts and uncertainties in complex systems asks for new modeling formal methods and software tools that can deal with managing such collections of heterogeneous and dynamical interacting entities. The purpose of this paper is to present in detail the implementation of a new software modeling platform which is based on the hLARM (Hybrid Logic-Algebraic Relational Modeling) formalism which aims to alleviate the process of designing and managing such systems. The development of the platform is thoroughly analyzed from its conceptual background and up to the algorithms which support all the inference procedures involved.

Key-words: Complexity; modeling; platform; relational modeling; trust factors.

1. Introduction

The era of connectivity in which the machines connect to each other and collaborate with the human operators, opened the way to a new stage in the engineering of complex systems as heterogeneous and hierarchically and heterarchically organized systems [1]. New fields of science such as Neuroscience, Ecology or Social Networks, call for a new approach to manage the complexity [2]. Cyber-Physical Systems, the Internet of Things and Services, Socio-technical Systems led to development of new software platforms for design, analysis, verification and validation. The use of classical mathematical models for the characterization of such complex systems is limited and ineffective, requiring to consider other categories of heuristic, relational, logical and algebraic representations. The paper presents a new software modeling platform based on the hLARM (Hybrid Logic-Algebraic Relational Modeling) formalism which was introduced and thoroughly formalized in [3]. The formalism is based on relational models [4], logic, probabilities, numerical information and network representations which provides an intuitive, highly

flexible and powerful tool to define, analyze and infer behaviors, facts and uncertainties in complex systems.

In the research community, there were developed in recent years several software platforms for modeling various aspects in complex systems which rely on specific formal support. For hybrid systems, using the formalism of hybrid programs [5], it was developed *KeYmaera X* [6]. It is web based and provides means to define hybrid models consisting of differential equations and logical expressions which can be analyzed against various properties, constraints or theorems using the embedded solver. Modeling in a graphical manner various complex dynamical systems consisting of a wide range of mechanical, hydraulic, electrical, thermal or chemical processes [7] can be achieved using *Modelica* [8] or *Simulink* [9]. Behind the block diagram representation there are differential equations and solvers. For implementing and simulating multi-agent systems *NetLogo* [10] can be used. The emergent behavior is achieved here by the interaction of multiple classes of agents, each having multiple instances. The autonomous behavior of each agent class is defined using a dedicated programming language [11]. The *Ptolemy II* modeling platform [12] implements the actor-based formalism which focuses on synchronization and scheduling in complex concurrent systems. Here there are defined multiple semantics which control how the entities defined in the platform as blocks with links communicate and perform their internal computation [13]. Based on the *SysML* standard [14] for designing and defining systems in general [15], some companies developed software platforms that asses the modeling process or provide assistive tools for checking constraints, consistency, etc. Modelio [16] is an example of such a platform.

In this paper, the implementation of LARM is presented in detail. A conceptual description of the hLARM formalism is presented in Section 2. The architecture of the platform is outlined in Section 3 and its implementation in Section 4. The algorithms employed in performing the procedures of the platform are formalized in Section 5. Finally, conclusions are drawn in Section 6.

2. Brief Description of the hLARM Modeling Formalism

The hLARM formalism addresses the problem of describing workflows in complex systems. Such systems comprise of multiple autonomous entities which are interacting in a network-like structure and have their own internal structure and behavior. The outcome of the modeling is the analysis of the global behavior which is emerging from the interaction of the entities. Additionally, it is useful for designing specific external influence which can be exerted to drive the global behavior towards a desired one.

Each entity involved is abstracted by a model and the entire context of the complex system by an environment of models with links (as depicted in Fig. 1). Each model has parameters and attached attributes which can be inputs, internal or outputs. The parameters (denoted as IOs) define physical quantities, states or components which are involved either in the interaction of the models, or in the internal behavior. They can be treated logically (in terms of True or False) based on their attributes which define specific quality or criteria the parameter can hold, or can be treated numerically. From a formal perspective, the parameters (input, internal and output) are denoted as u_i, w_i and y_i, and their attached attributes as $\alpha^j_{u_i}, \alpha^j_{w_i}, \alpha^j_{y_i}$.

The internal behavior of each model is defined using logical rules (predicates) with attached trust factors (probabilities) embedding any attribute of the model. The set of the rules is denoted as F and the attached trust factors as Π, having the following formal definitions:

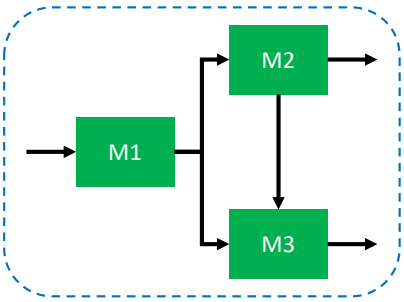

Fig. 1. An example of a modeling environment with three interacting entities abstracted by models and links.

$$F(\alpha) = F(\alpha_u, \alpha_w, \alpha_y) = \begin{bmatrix} F^1(\alpha_u, \alpha_w, \alpha_y) \\ F^2(\alpha_u, \alpha_w, \alpha_y) \\ \vdots \end{bmatrix}, \quad \Pi = \begin{bmatrix} p(F^1) \\ p(F^2) \\ \vdots \end{bmatrix} \tag{1}$$

Input and output domains can be defined both at the model level and at the environment level which interact with the input and output parameters and attributes. The input domains are denoted as D_u while the output ones as D_y and contain logical predicates with attached trust factors ($F_u(\alpha_u), \Pi_u$ and $F_y(\alpha_y), \Pi_y$) for logical parameters and probability density functions ($p_{u_k}(u_k)$ and $p_{y_k}(y_k)$) for numerical parameters.

Starting from this concept of interacting models with input-output interface, internal behavior and domains, three inference procedures are defined:

- *Analysis*: given an input domain for free input parameters of the environment, determine the corresponding output domain such that all logical predicates are satisfied and all the trust factors are consistent.
- *Synthesis*: given a desired output domain for free output parameters of the environment, determine the corresponding input domain.
- *Structural equivalence*: given an environment with models, determine one model that has an equivalent input-output behavior.

More details regarding the hLARM formalism can be found in the original paper [3].

3. Modeling Platform Architecture

At the core of any modeling platform there are two complementary classes of components: Internal ones which deal with all the functional implementation and the Graphical User Interface (GUI) which makes available for the user all the resources of the platform. The internal components are comprised of various resources which are realizations of the entities from the modeling formalism and various algorithms with specific purpose performing inference procedures over the resources. It is therefore very important to highlight that the functional core of a modeling

platform is the **Concept** - **Resource** - **Procedure** trio. The GUI has two main objectives, to construct equivalent graphical representations of the resources and to take commands from the user in order to trigger specific actions to the internals of the platform.

In developing the LARM platform we adopted such an approach and its conceptual architecture is highlighted in Fig. 2. The resources of the platform are formal entities acquired from the hLARM conceptual framework: models, parameters, attributes, conversion equations, rules with attached trust factors, links, logical domains, numerical domains.

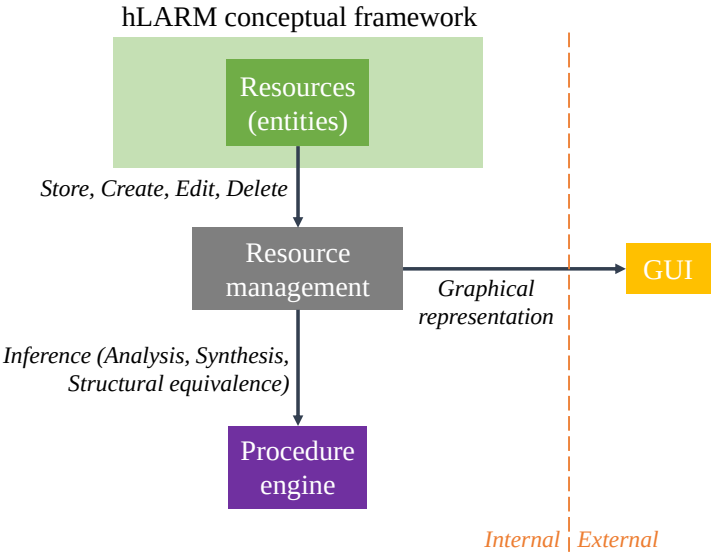

Fig. 2. Conceptual architecture of the LARM modeling platform.

The management involves storing, creating, editing and deleting resources using dedicated data structures that are presented in the next section. For each resource stored in the modeling environment, a specific graphical representation is rendered in the GUI. More details are also given in the next section. The Procedure engine runs the algorithms implemented to perform all the inference mechanisms required for the analysis, synthesis and structural equivalence procedures (domain splitting, domain combination, domain conversion, logical inference and probabilistic inference).

As depicted in Fig. 3 the two main components of the platform are the LARM terminal and the LARM UI. The former is a standalone application which implements the resource management and the procedure engine. It runs inside the Interpreter and facilitates the interaction with the user through menus in the OS Command Prompt / Terminal. The LARM UI is a web application hosted in a local Web Server which provides graphical representations of all the resources in the platform and assistance for navigation. The graphical elements and the content of the interface are dynamically generated by the LARM terminal which are rendered by a Web Browser. The interaction between the LARM terminal and the UI is achieved by the user with the help of an Index mechanism which is presented in the next section.

For the implementation of the platform, the software technologies used are depicted in Fig. 4. The LARM terminal is implemented in Python [17] and uses the followind packages: SymPy [18] (for symbolic and boolean algebra calculus), NumPy [19] (for matrix management and linear

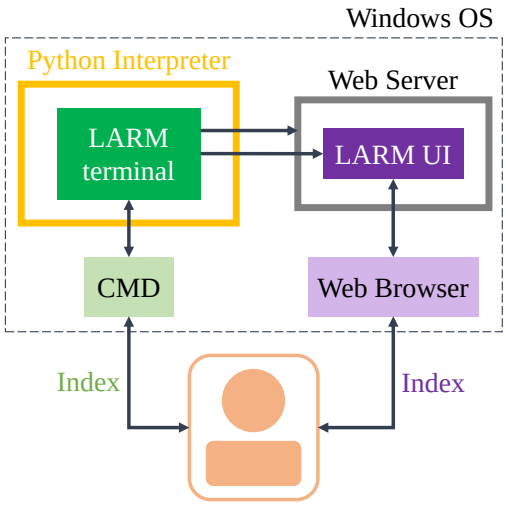

Fig. 3. The sotware architecture of the LARM modeling platform.

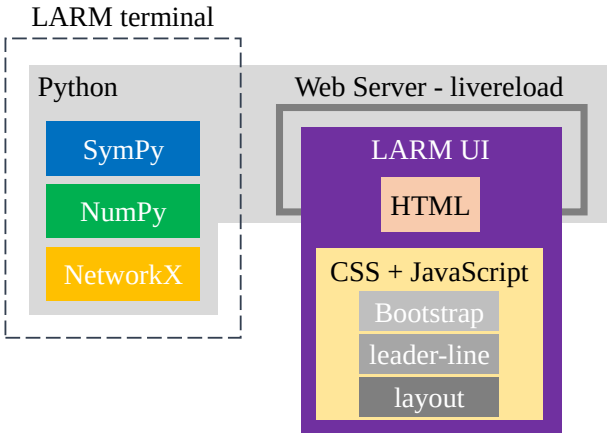

Fig. 4. The outline schematic of the software technologies used for developing the LARM platform.

algebra calculus), NetworkX [20] (for the UI layout algorithm).

The LARM UI is hosted using a Python livereload server [21] which is initialized by the LARM terminal. It triggers the browser to reload every time it detects a change of the web pages, which makes the UI interactive. For its development HTML, CSS and JavaScript languages were used, together with Bootstrap [22], leader-line [23] for rendering connections and the layout algorithm which is presented in the next section.

4. Platform Implementation

The LARM platform provides means to address the modeling in a concurrent and incremental manner. Multiple users can contribute to the development of a modeling environment where each sub-part can have different levels of details. There may be both incipient ideas tested and mature behaviors described in the same environment and the inference procedures are able to address this context. The following four levels of detail are available:

1. Define models and external links between them
2. Define internal attribute links which for this level behave as logical implications
3. Define model behavior using logical predicates with attached trust factors
4. Define numerical parameters with higher modeling precision for real scenarios

4.1. LARM terminal

The resources of the platform are hierarchically distributed such that the modeling environment can have multiple models, each model can have multiple IOs and each IO can have multiple attributes. The same distribution is available for connections where links between models can include multiple links between IOs which can further include multiple links between attributes. In managing (creating, editing, deleting) the resources and links, considering their tree-like structure, multiple constraints are established such that the modeling environment is consistent.

The LARM terminal is implemented using 15 classes (depicted in Fig. 5) which are interrelated considering the hierarchy of the resources. For easily identifying each resource, an unique alphabetical index is cryptographically generated. For highly efficient searching, the resources are stored in dictionary data structures in the format:

$$\{\text{'index'} : \text{object related to the resource}\} \tag{2}$$

The *terminal* is implemented around a state machine with various menus which assists the user to manage and navigate through the resources, to define links, to file save, import and combine environments and to define logical and numerical domains for analysis, synthesis and structural equivalence inferences. The main menu of the terminal is depicted in Fig. 6.

The *laEnv* class abstracts the modeling environment which contains a dictionary with all the models and three graphs for model, IO and attribute links. The *models* class has an index, a name, a dictionary with logical predicates with attached trust factors for defining its internal behavior (abstracted by the *modRule* class) and a dictionary of IOs. The *ios* class has an index, a name, a type (if it's input, internal or output) a symbol used within the conversion equations or the numerical domains and a dictionary of attributes. The *atts* class has an index, a name, a symbol used within the logical predicates or the logical domains and a domain that describes the interval where the conversion equation makes the attribute True. The *graph* class has a dictionary of links while the *link* class contains and index, the indexes of the source and destination resources and the index of the parent link. During the analysis, synthesis or structural equivalence inference procedures, the intermediary domains between models are stored as a dictionary in the *envContext* class. A domain is abstracted with the *laDomain* class having a list of logical predicates with attached trust factors and a dictionary of numerical domains. The *envInterface* class is used during the domain conversion and propagation procedures and defines how the current model is

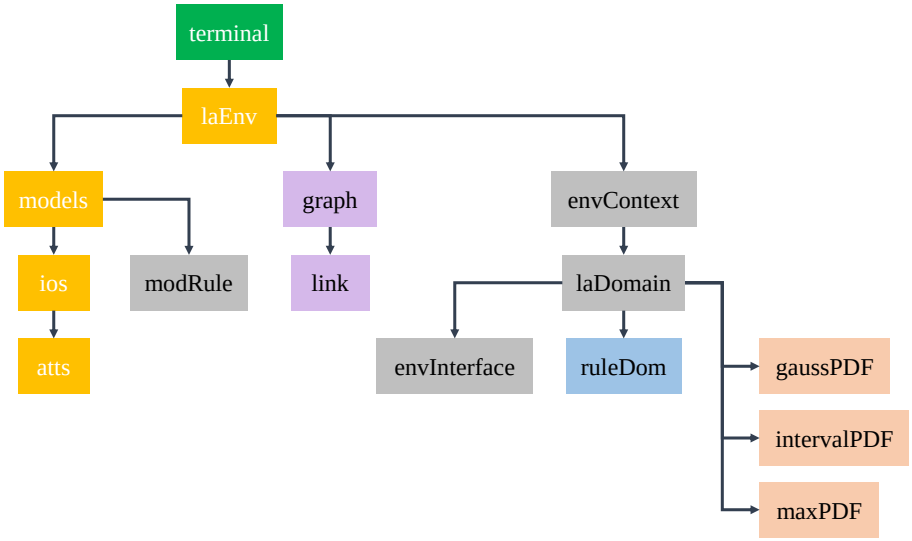

Fig. 5. The tree of classes developed for the LARM platform.

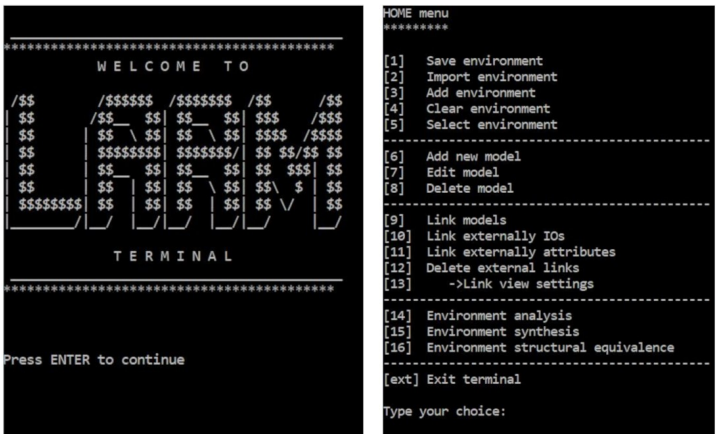

Fig. 6. Screen capture of the welcome screen and main menu of the terminal from the LARM platform.

connected to preceding and following models, which types of links are used and what IOs and attributes are involved. A logical predicate is abstracted using the *ruleDom* class which contains a symbolic logical expression and the value of the trust factor. Three types of numerical domains (Probability Density Functions) are considered and implemented: a Gaussian distribution (*gaussPDF* class), a constant distribution in a range (*intervalPDF* class) and a union of interval distributions (*maxPDF* class).

Te modeling environment can be saved to a *.larm* format file and later imported in the platform using the pickle python package [citare]. Also, multiple files can be simultaneously imported in the platform which makes the corresponding environments to merge.

4.2. LARM UI

Each resource in the platform has a dedicated graphical representation implemented using HTML, CSS and Bootstrap. The model (Fig. 7-a) is represented as a green rectangle with the name as a label in the upper left corner and the list of logical predicates with attached trust factors on the bottom. It encapsulates all the resources (IOs, Attributes, internal connections) it owns. A tooltip (Fig. 7-h) with additional information regarding the model appears when hovering the label with the mouse. The IOs are represented as a rectangle with rounded corners having a name label on the upper side and the Attributes it owns on the bottom (Fig. 7-b-f). The colors used are considering the type of the IO: inputs are using yellow, internal parameters are using blue and outputs are using magenta. An input which is treated numerically has an orange border around its label (Fig. 7-c) and an output one has a purple border (Fig. 7-f). Similar with a model, when hovering the IO label a tooltip with additional information is shown (Fig. 7-i). An attribute is represented using a grey elipse (Fig. 7-g) which contains the name and the symbol of the attribute which is used inside the logical predicates. In Fig. 7-j is depicted the tooltip for an attribute which belongs to an IO which is treated numerically, thereby, the numerical to logical conversion domain is shown.

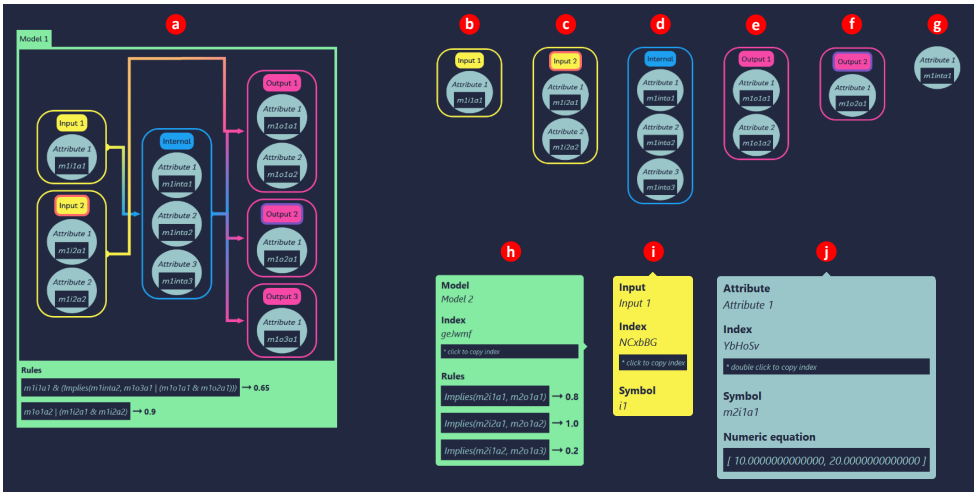

Fig. 7. A map of all the graphical representation of the resources that can be defined in the LARM platform.

The links are implemented using the leader-line library. There are three types of links:

- Model links - external links between models rendered in green color (Fig. 8-a)
- IO links - external and internal links between IOs rendered in gradient color considering the color of the starting and ending element (Fig. 8-b)
- Attribute links - external and internal links between attributes rendered in grey color (Fig. 8-c)

Considering that usually the number of links in an environment may be large and visualizing the connections between resources can be cumbersome, a focus mechanism was developed.

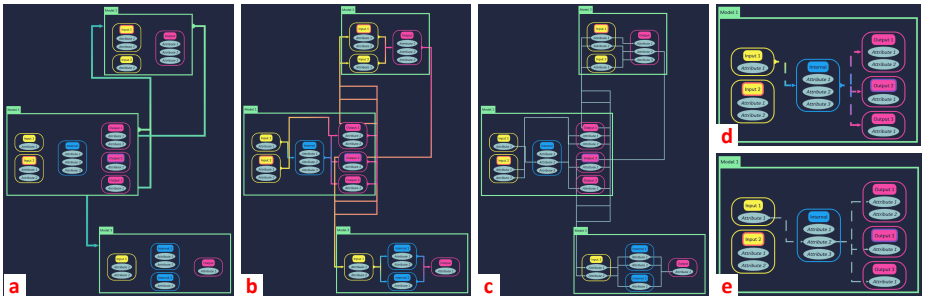

Fig. 8. The graphical representation of the model, parameter and attribute links and their animations.

Thus, when clicking on a resource (model, IO, attribute) all the links disappear except the inbound and outbound ones, which are starting to play a flow animation using a dotted stroke (Fig. 8-d,e).

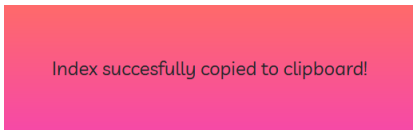

Fig. 9. Pop-up window shown when the user duple clicks a resource in order to copy its index to clipboard.

When interacting with the LARM terminal, each operation of selection for editing / deleting or linking requires to input the unique index of the resources involved. In order to ease this operation, when double clicking a resource, its index is automatically copied to clipboard (Fig. 9) which can be easily pasted afterwards.

4.3. Layout algorithm

The purpose of the layout algorithm is to position all the graphical elements of the LARM UI in an optimal manner such that for large modeling environments the user can perceive all the details. Because the resources of the platform are interconnected, the layout algorithm has the same goal as graph visualization techniques [24] [25]: to position the models in order to avoid overlapping and to achieve optimal link routing.

Layout algorithms for visualizing graphs have an important role in emphasizing structural properties like node and link rankings or clusters. Such information is used for analyzing interactions between entities in large-scale systems. Among the developed methods, the most popular ones are the orthogonal layout algorithms [26] [27] and the force-directed ones [28].

The leader-line library is not providing any mean to control the routing of the links and because of this, the objective here is only to position the models in an optimal manner. A first positioning is achieved by a force-based kamada-kawai algorithm included in the NetworkX package [citare] which is applied for the graph of model links. The space of coordinates provided by this algorithm is $[-1, 1]$ and the size of the models is also not taken into account. Thereby, the LARM UI layout algorithm implemented in JavaScript is performing next a coordinate translation pro-

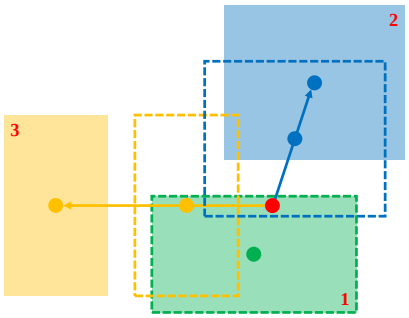

Fig. 10. A graphical representation of the expansion algorithm implemented to translate the coordinates provided by the kamada-kawai algorithm to pixel values.

cedure to pixel values, which is also considering the size of the models and if they are connected or not. The un-linked models are positioned in a grid on top of the interface. The coordinates of the other ones are expanded outwards from the center of mass of the group of linked models in order to ensure that none of them overlap. The algorithm is started from the model which is closest to the center of mass (which is kept in place) and follows in a sequence up to the furthest one. The procedure is depicted in Fig. 10 where the red dot represents the center of mass of the group of linked models having the initial coordinates (depicted as dotted rectangles) provided by the kamada-kawai algorithm, while the solid stroke rectangles represents the final position after the expansion. This approach is managing to preserve in the final layout the relative positions of the models provided by the initial kamada-kawai algorithm. The mathematical formulation of the procedure is given in Algorithm 1

5. hLARM Algorithm Implementation

In order to implement the inference procedures of hLARM formalized in Algorithms 1-5 from [3], custom methods were developed in the LARM terminal classes which rely on external resources and solvers detailed in the following.

The σ operator is implemented using the symbolic SAT solver from the logical module in the SymPy package, which is based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [29]. In this implementation the logical contradictions in the environment can be detected when the solution of the SAT algorithm is False and is further propagated from one model to another. This leaves a clear trace of the point where the problem originated. The conversion operator γ is implemented using the POSform (Product of Sums) algorithm from the SymPy package, which is based on the Quine-McCluskey algorithm [30] [31] [32]. In this implementation, the logical decoupling (non causal behaviors) can be identified when the outcome of the POSform is True, signifying that all the previous logical constraints have no effect. The filter operator Φ is implemented using standard set theory over Python dictionaries. The probabilistic inference is using standard matrix calculus implemented using the NumPy package. The environment level procedures are developed around a recursive depth-first search transversal algorithm which was adopted for processing the graph of models.

In the following, the main steps of the algorithms are detailed, while the classes of the resources involved are indicated.

Algorithm 1 LARM UI layout algorithm

Input:

(w_k^u, h_k^u) - the size (width and height) of un-linked models
(x_k^l, y_k^l) - the initial coordinates of linked models
(w_k^l, h_k^l) - the size (width and height) of linked models

Output:

(x_k^u, y_k^u) - the final coordinates (of the top left corner in pixels) of un-linked models
(x_k^l, y_k^l) - the final coordinates (of the top left corner in pixels) of linked models

▷ Position un-linked models

$(x_1^u, y_1^u) = (100, 100)$
for $k = 2, \ldots, n$ **do**
    $(x_k^u, y_k^u) = (x_{k-1}^u + w_{k-1}^u + 100, 100)$
end for
$(x_c, yc) = \left(\frac{\sum_{k=1}^{m} x_k^l \cdot w_k^l \cdot h_k^l}{\sum_{k=1}^{m} w_k^l \cdot h_k^l}, \frac{\sum_{k=1}^{m} y_k^l \cdot w_k^l \cdot h_k^l}{\sum_{k=1}^{m} w_k^l \cdot h_k^l} \right)$ ▷ Compute center of mass coordinates
for $k = 1, \ldots, m$ **do**
    $d_k = \sqrt{(x_k^l - x_c)^2 + (y_k^l - y_c)^2}$ ▷ Compute the distances to the center of mass for each model
end for
$(x_k^l, y_k^l, w_k^l, h_k^l) = sort(x_k^l, y_k^l, w_k^l, h_k^l; d)$ ▷ Sort the vectors of coordinates and sizes in ascending order by the distance
for $k = 2, \ldots, m$ **do**
    $(x, y) = (x_k^l + \frac{w_k^l}{2}, y_k^l + \frac{h_k^l}{2})$ ▷ Compute center of mass for current model

▷ Compute target coordinates

    if $x == x_c$ **then**
        $x_T = x$
    end if
    if $y == y_c$ **then**
        $y_T = y$
    end if
    if $x < x_c$ **then**
        $x_T = (\min_{i=1,\ldots,k} x_i^l) - w_k^l - 200$
    end if
    if $x > x_c$ **then**
        $x_T = (\max_{i=1,\ldots,k} x_i^l + w_i^l) + 200$
    end if
    if $y < y_c$ **then**
        $y_T = (\min_{i=1,\ldots,k} y_i^l) - h_k^l - 100$
    end if
    if $y > y_c$ **then**
        $y_T = (\max_{i=1,\ldots,k} y_i^l + h_i^l) + 100$
    end if

▷ Compute new coordinates for current model

    if $x_T - x_k^l \geq y_T - y_k^l$ **then**
        $(x_k^l, y_k^l) = (x_T, \frac{(x_T - x_k^l) \cdot (y_k^l - y_c)}{(x_k^l - x_c)} + y_k^l)$
    else
        $(x_k^l, y_k^l) = (\frac{(y_T - y_k^l) \cdot (x_k^l - x_c)}{(y_k^l - y_c)} + x_k^l, y_T)$
    end if
end for

5.1. Algorithms for analysis procedures

The environment level analysis procedure (Algorithm 2) and the model analysis procedure (Algorithm 3) are implemented as methods in the *laEnv* class.

Algorithm 2 Environment level analysis procedure

1: Determine the input interface of the environment (*laEnv* method)
2: Prompt the user to create the input domain for free inputs (*laDomain* method)
3: Clear the context in the environment
4: Add the input domain to the context (*envContext* method)
5: Split the input domain to models with free inputs (*laDomain* method)
6: Save all the resulting domains to the context (*envContext* method)
7: **for** each model with free inputs **do**
8: Call the model analysis procedure (*laEnv* method)
9: **end for**
10: Determine the output interface of the environment (*laEnv* method)
11: Gather from the context all output domains of models with free outputs (*envContext* method)
12: Logically simplify the resulting output domain (*laDomain* method)
13: Add the resulting domain to the context (*envContext* method)
14: Print to the terminal the resulting output domain (*laDomain* method)
15: Save the context to file (*envContext* method)

Algorithm 3 Model analysis procedure

1: Determine the model input interface and free input links (*laEnv* method)
2: Check if the model has rules or internal attribute connections
3: Check if all the output domains of models from the input interface are available in the context
4: Combine all input domains (*laDomain* method)
5: Transfer from left to right the resulting input domain (*laDomain* method)
6: Convert numerical domains to logical (*laDomain* method)
7: Add to the input domain the internal logical predicates (*laDomain* method)
8: Add implications for attribute links without corresponding rules
9: Rename the symbols in rules for loop links (*laDomain* method)
10: Determine the model output interface and free output links (*laEnv* method)
11: Solve the rules and split the resulting output domain in multiple domains for each successive model and free output patameter (*laDomain* method)
12: Save all the domains in the context (*envContext* method)
13: **for** each successive model **do**
14: Call the model analysis procedure (*laEnv* method)
15: **end for**

5.2. Algorithms for synthesis procedures

The environment level synthesis procedure (Algorithm 4) and the model synthesis procedure (Algorithm 5) are implemented as methods in the *laEnv* class.

Algorithm 4 Environment level synthesis procedure

1: Determine the output interface of the environment (*laEnv* method)
2: Prompt the user to create the output domain for free outputs (*laDomain* method)
3: Clear the context in the environment
4: Add the output domain to the context (*envContext* method)
5: Split the output domain to models with free outputs (*laDomain* method)
6: Save all the resulting domains to the context (*envContext* method)
7: **for** each model with free outputs **do**
8: Call the model synthesis procedure (*laEnv* method)
9: **end for**
10: Determine the input interface of the environment (*laEnv* method)
11: Gather from the context all input domains of models with free inputs (*envContext* method)
12: Logically simplify the resulting input domain (*laDomain* method)
13: Add the resulting domain to the context (*envContext* method)
14: Print to the terminal the resulting input domain (*laDomain* method)
15: Save the context to file (*envContext* method)

Algorithm 5 Model synthesis procedure

1: Determine the model output interface and free output links (*laEnv* method)
2: Check if the model has rules or internal attribute connections
3: Check if all the input domains of models from the output interface are available in the context
4: Combine all output domains (*laDomain* method)
5: Transfer from right to left the resulting output domain (*laDomain* method)
6: Convert numerical domains to logical (*laDomain* method)
7: Add to the output domain the internal logical predicates (*laDomain* method)
8: Add implications for attribute links without corresponding rules
9: Rename the symbols in rules for loop links (*laDomain* method)
10: Compute the set of solutions that satisfy $F \wedge F^y$ and does not satisfy $F \wedge \neg F^y$
11: Determine the model input interface and free input links (*laEnv* method)
12: Split the resulting input domain in multiple domains for each preceding model and free input parameter (*laDomain* method)
13: Save all the domains in the context (*envContext* method)
14: **for** each preceding model **do**
15: Call the model synthesis procedure (*laEnv* method)
16: **end for**

5.3. Algorithms for structural equivalence procedures

The environment level structural equivalence procedure (Algorithm 6) is implemented as a method in the *laEnv* class. The model structural equivalence procedure is similar to the model analysis procedure (Algorithm 3).

Algorithm 6 Environment level structural equivalence procedure

1: Determine the input and output interface of the environment (*laEnv* method)
2: Clear the context in the environment
3: **for** each model with free inputs **do**
4: Call the model structural equivalence procedure (*laEnv* method)
5: **end for**
6: Gather from the context all output domains of models with free outputs (*envContext* method)
7: Logically simplify the resulting output domain (*laDomain* method)
8: Add the resulting domain to the context (*envContext* method)
9: Create a new environment
10: Create a new model
11: Add to the new model as input and output parameters the free inputs and outputs of the initial environment
12: Add as internal rules for the new model the resulting logical output domain of the above inference procedure

5.4. Implementation of the internal probabilistic inference

The probabilistic inference is performed using the following equation (Definition 8 from [3]):

$$\Pi_y = \Pi'^T \cdot (V' \cdot V'^T)^{-1} \cdot V' \cdot V_y^T \tag{3}$$

Computing the elements of the V matrix is performed on $F_u(\alpha_u)$, $F(\alpha)$ and $F_y(\alpha_y)$ sets of logical predicates using the σ operator. For this purpose, an auxiliary attribute f_i is defined for each logical predicate and the following rule is built:

$$F_V = \bigwedge_i ITE(f_i, F_i, \neg F_i) \tag{4}$$

where F_i are all the logical predicates from $F_u(\alpha_u)$, $F(\alpha)$ and $F_y(\alpha_y)$ and ITE is the conditional "If Then Else" function. The F_V logical predicate is than solved:

$$\bar{S}_V = \sigma | F_V \tag{5}$$

and the solution is further filtered after the set of auxiliary attributes f_i:

$$\bar{S}_f = \phi_{f_i} | \bar{S}_V \tag{6}$$

Each $V_{i,j}$ element of the V matrix has the truth value (1 or 0) of the f_i attribute from the j solution of the $\bar{S}_f$ set of solutions. In order to determine the linearly independent lines from the V' matrix the *SymPy* is used again.

6. Conclusions

The paper presents a new software platform for modeling heterogeneous complex systems based on the hLARM formalism (Hybrid Logic-Algebraic Relational Modeling) presented in detail in [3]. The architecture of the platform and the algorithms developed highlight a high degree of flexibility and real ease of use. The proposed platform is based on resources, procedures and a Graphical User Interface having two main objectives, to build equivalent graphic representations of the resources and to capture the user's commands.

The platform's resources are formal entities taken from the hLARM conceptual framework that give the platform simplicity of configuration and high adaptability for wide classes of heterogeneous complex systems.

The essential advantages of the developed platform is the reconfigurability of different entities modeled and parameterized according to the role and place within the structure of the complex system. The structural-functional integration and the dynamic behavior of the system are easily captured by using relational models, logic, probabilities, numerical information and network representations with efficient interfacing.

Acknowledgement. The results presented in this article have been funded by the Ministry of Investments and European Projects through the Human Capital Sectoral Operational Program 2014-2020, Contract no. 62461/03.06.2022, SMIS code 153735.

References

[1] S. THURNER, P. KLIMEK, and R. HANEL, *Introduction to the Theory of Complex Systems*. Oxford University Press, 2018.

[2] S. GU, F. PASQUALETTI, M. CIESLAK, Q.-K. TELESFORD, A.-B. YU, A.-E. KAHN, J.-D. MEDAGLIA, J.-M. VETTEL, M.-B. MILLER, S.-T. GRAFTON and D.-S. BASSETT, *Controllability of structural brain networks*, Nature Communications **6**, 2015, p. 8414.

[3] D. C. POPESCU and I. DUMITRACHE, *Knowledge representation and reasoning using interconnected uncertain rules for describing workflows in complex systems*, Information Fusion **93**, 2023, pp. 412–428.

[4] Z. BUBNICKI, *Modern Control Theory*, Springer-Verlag, Berlin, Heidelberg, 2005.

[5] A. PLATZER, *Logical Foundations of Cyber-Physical Systems*, Springer, Cham, 2018.

[6] *KeYmaera X: An aXiomatic Tactical Theorem Prover for Hybrid Systems*. Accessed: September 5, 2023 [Online]. Available: https://keymaerax.org/index.html.

[7] I. LIND and H. ANDERSSON, *Model based systems engineering for aircraft systems - how does modelica based tools fit?*, Proceedings of 18^{th} International Modelica Conference, Dresden, Germany. Linkoping Electronic Conference Proceedings **63**, pp. 856–864, 2011.

[8] *The Modelica Association*. Accessed: September 5, 2023 [Online]. Available: https://www.modelica.org/.

[9] *Simulink - Simulation and Model-Based Design*. Accessed: September 5, 2023 [Online]. Available: https://www.mathworks.com/products/simulink.html.

[10] *NetLogo Home Page*. Accessed: September 5, 2023 [Online]. Available: https://ccl.northwestern.edu/netlogo/.

[11] S. F. RAILSBACK and V. GRIMM, *Agent-Based and Individual-Based Modeling: A Practical Introduction*, Princeton University Press, 2011.

[12] *Ptolemy II Home Page*. Accessed: September 5, 2023 [Online]. Available: https://ptolemy.berkeley.edu/ptolemyII/index.htm.

[13] C. PTOLEMAEUS, Ed., *System Design, Modeling, and Simulation Using Ptolemy*, Ptolemy.org, 2014.

[14] *SysML Open Source Project: What is SysML? Who created SysML?*. Accessed: September 5, 2023 [Online]. Available: https://sysml.org/index.html.

[15] E. PALACHI, C. COHEN, and S. TAKASHI, *Simulation of cyber physical models using SysML and numerical solvers*, Proceedings of 2013 IEEE International Systems Conference, Orlando, FL, USA, 2013, pp. 671–675.

[16] *Modelio Open Source - UML and BPMN free modeling tool*. Accessed: September 5, 2023 [Online]. Available: https://www.modelio.org/index.htm.

[17] *Welcome to Python.org*. Accessed: September 5, 2023 [Online]. Available: https://www.python.org/.

[18] *SymPy*. Accessed: September 5, 2023 [Online]. Available: https://www.sympy.org/en/index.html.

[19] *NumPy*. Accessed: September 5, 2023 [Online]. Available: https://numpy.org/.

[20] *NetworkX - NetworkX documentation*. Accessed: September 5, 2023 [Online]. Available: https://networkx.org/.

[21] *LiveReload Web Server*. Accessed: September 5, 2023 [Online]. Available: https://github.com/lepture/python-livereload.

[22] *Bootstrap · The most popular HTML, CSS, and JS library in the world*. Accessed: September 5, 2023 [Online]. Available: https://getbootstrap.com/.

[23] *LeaderLine*. Accessed: September 5, 2023 [Online]. Available: https://anseki.github.io/leader-line/.

[24] H. GIBSON, J. FAITH and P. VICKERS, *A survey of two-dimensional graph layout techniques for information visualisation*, Information Visualization **12**(3–4), 2013, pp. 324–357.

[25] Y. HU, *Algorithms for visualizing large networks*, Combinatorial Scientific Computing **5**(3), 2011, pp. 180–186.

[26] J. SUN, *Automatic Orthogonal Graph Layout*, Bachelor's Thesis, Hamburg University of Technology, Hamburg, Germany, 2007.

[27] K. FREIVALDS and J. GLAGOĻEVS, *Graph compact orthogonal layout algorithm*, in Combinatorial Optimization: Third International Symposium, ISCO 2014, Lisbon, Portugal, Springer, pp. 255–266, 2014.

[28] Y. F. HU, *Efficient and high quality force-directed graph drawing*, The Mathematica Journal **10**, pp. 37–71, 2005.

[29] M. DAVIS and H. PUTNAM, *A computing procedure for quantification theory*, Journal of The Acm **7**, 1960, pp. 201–215.

[30] W. V. QUINE, *The problem of simplifying truth functions*, The American Mathematical Monthly **59**(8), 1952, pp. 521–531.

[31] W. V. QUINE, *A way to simplify truth functions*, The American Mathematical Monthly **62**(9), pp. 627–631, 1955.

[32] E. J. McCluskey, *Minimization of boolean functions*, The Bell System Technical Journal **35**(6), 1956, pp. 417–1444.