A Graph Matching Algorithm to Extend Software Wise Systems with Human Semantic

Abdelhafid Dahhani
LISTIC
Université Savoie Mont Blanc
Annecy, France
email: abdelhafid.dahhani@univ-smb.fr
0000-0001-6314-662X

Ilham Alloui
LISTIC
Université Savoie Mont Blanc
Annecy, France
email: ilham.alloui@univ-smb.fr
0000-0002-3713-0592

Sébastien Monnet
LISTIC
Université Savoie Mont Blanc
Annecy, France
email: sebastien.monnet@univ-smb.fr
0000-0002-6036-3060

Flavien Vernier
LISTIC
Université Savoie Mont Blanc
Annecy, France
email: flavien.vernier@univ-smb.fr
0000-0001-7684-6502

***Abstract*—Wise systems refer to distributed communicating software objects, which we termed Wise Objects, able to autonomously learn how they are expected to behave and how they are used. Wise Objects are designed to be associated either with software or physical objects (e.g., home automation) to adapt to end users while demanding little attention from them. This last requirement obeys to the principle of calm technology introduced by Mark Weiser and John Seely Brown in 1995. Wise Objects are endowed with autonomous computing capabilities as they implement the notion of IBM's 4 state loop Monitor-Analyze-Plan-Execute over a shared Knowledge. However, they suffer from a lack of semantic, which prevents them from communicating effectively with a human. The work presented in this paper aims at extending Wise Objects with the ability to use human semantic to communicate with a user. Construction of such systems requires at least two views: (i) a conceptual view relying on knowledge given by developers to either control or specify the expected system behavior; and (ii) an auto-generated view acquired by wise systems during their learning process. The problem is that, while a conceptual view is understandable by humans (i.e., developers, experts, etc.), a view generated by a software system contains mainly numerical information with mostly no meaning for humans. In this paper, we address the issue of how to relate both views using two state-based formalisms: Input Output Symbolic Transition Systems for conceptual views and State Transition Graphs for views generated by the wise systems. Our proposal is to extend the generated knowledge with the conceptual knowledge using a matching algorithm founded on graph morphism. Target results are twofold: (i) make wise systems' generated knowledge understandable by humans, (ii) enable human evaluation of wise systems' outputs. To illustrate the overall process, the construction of two samples of graph matching on a roller shutter and a light bulb are considered.**

Keywords*—*statecharts; monitoring systems; adaptive system and control; knowledge-based systems; discrete-event systems; graph matching; semantic.

I. Introduction

In recent years, software technology has exponentially undergone a huge evolution increasing the development of intelligent applications using AI models and techniques, in particular machine learning techniques. Examples of AI-based systems are home-automation and software network traffic analysis. Artificial intelligence (AI) and software engineering (SE) are old interdependent and mutually beneficial fields that came into the spotlight with the advent of deep learning. The challenges for SE are linked to modeling, implementing and testing software and systems that integrate AI. When AI provides software with ability to adapt to their environment, designing such software requires dedicated approaches and tools to manage this ability leading to a non predictable software behavior. Hence, the idea underlying the Wise Systems (WS) [1] is to provide developers with software support, to help them design and build intelligent systems and applications. Indeed the availability of machine learning libraries and off-the-shelf solutions sometimes gives the illusion that developing AI-based software applications is easy, but the reality is that developing viable and trusted AI systems requires significant effort [2]–[5] and combination of AI and SE.

Wise systems refer to distributed communicating software objects, which we termed Wise Objects (WOs), able to autonomously learn how they are expected to behave and how they are used. A desirable feature of a WS is self-adaptation: the WS should be able to autonomously adapt according to their use. It can be seen as a particular Multi-Agent System [6][7] that monitors only its internal changes and does not directly observe its external environment. Concretely, a WO is a piece of software able to monitor itself: the way it is used and the way it could be used (through introspection). The main specificity of a WO is that it is able to learn about itself in an autonomous way: it monitors its method invocations and their impact, it can also simulate method invocations to envision possible use and explore/discover new states. A method implements a service or functionality to be provided by a WO Then, the collected monitoring data can feed a learning process to be able to determine usual and unusual

2023, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

behavior (for instance), this learning process is implemented by experts through plugins [8]. Let us note that a plugin is software that adds new abilities or extends existing ones. We have developed several plugins for WOs, e.g., Analyzers, Planners, Graph Matchers, AI Models, etc. An example is a home-automation system that collects someone's behavior within a place and analyzes it to be able to act "silently" when necessary. Such systems should require the minimum attention from their end users while being able to adapt to changes in their behavior.

To meet those requirements and as the development of such systems is non-trivial, we developed an object based framework named Wise Object Framework [9] (WOF) to help developers design, deploy and evolve WSs. Generally, knowledge in AI-enabled systems can be provided according to two ways: describing a priori the arrangement of activities to be performed by the system, or, letting the system acquire the required knowledge using learning mechanisms.

In the former case, ontologies and/or scenarios are usually used to describe the arrangement of activities to achieve a goal as in [10][11]. In [10], functional behavior as well as interoperation of system entities are described a priori using state-diagrams. Reference [11] goes a step forward by combining ontologies to design ambient assisted living systems with specifications based on logic and analyzers to check in logic clauses before system deployment to create relevant scenarios. In those approaches, the end user is at the heart of the scenario creation process, as described in [12][13]. *In the second case*, knowledge is provided by the AI-enabled system in representations and views not necessarily understandable by humans. This relates to the wide problem of comprehensibility of AI and to the distance between the business domain and technological domain views [14].

In this context, the WO acquires by itself knowledge about its capabilities – services to be provided – and its use to moderate attention from the end-users [15] (Calm technology [16]). Mark Weiser and John Seely Brown [17] describe calm technology as "that which informs but does not demand our(users) focus or attention". The WO also analyzes this knowledge to generate new one. As a result, it produces a State Transition Graph (STG) of the WO behavior. We consider STGs as the most natural way to model system dynamics. More precisely, this graph is built by iteration, i.e., step-wise construction, during a process called introspection. This process is launched during a phase called dream phase in which the WO discovers all its states (configurations) [18]. The downside of an STG generated by a WO is that numeric data provided has no meaning for humans. In the literature [19][20], other graphs like Input Output Symbolic Transition Systems (IOSTSs) are often used by developers/experts to model the behavior of systems to manage them using oracle or controller synthesis. Since this type of graph enables conceptual models understandable by humans, it can increase the knowledge of WOs and bring semantic to STGs.

This paper extends [1], which consists in enhancing the generated knowledge with the conceptual knowledge using a matching algorithm based on graph morphism [21][22]. This provides the ability to make WSs' generated knowledge understandable by humans and to enable human evaluation of WSs' outputs. Explicitly, the contribution presented in this paper attempts to relate both views, consequently enabling machine-human communication: (a) a conceptual view relying on knowledge given by developers to either describe or control the system behavior, and, (b) behavior-related knowledge acquired during WS's learning process. In this way, we use two state-based formalisms:

- STGs for representing behavior-related knowledge generated by the WSs.
- IOSTSs for modelling conceptual views of developers/experts,

The rest of the paper is organized as follows. Section II is dedicated to the context, it presents the basic idea, describes the architectural overview and gives the definition of important terms. Section III presents STG and IOSTS formalisms and illustrates them through examples. Finally, Section IV presents our graph matching algorithm and the construction of two samples of graph matching, one on a roller shutter and the other one on a light bulb to illustrate the algorithm. Section V is devoted to related work before concluding the paper in Section VI.

II. Context & problematic

Reusability and evolution in a wide architectures are two major problems faced when developing applications based on AI. The main challenge is then to provide support for the development of AI-based applications in a way similar to the development of "classical" software using software engineering methods [23][24], tools and languages [25]. We sketch in this section an overall view of how we designed the WO concept and the WOF to support such evolution.

As software systems play an important role in our daily life, their usage may vary depending on the end user and may evolve in time. Our research work is centred on AI-based applications, which are able to acquire data from their environment, to manipulate and to analyze them either to help users take decisions or to autonomously adapt their behavior to users' needs resulting the WOs concept.

A. Basic idea & definitions

The basic idea underlying the WO concept is to give a software entity (object, component, subsystem, etc.) the core mechanisms for learning behavior through introspection and analysis. Our aim is to go further by enabling software to execute "Monitoring", "Analyze", "Plan" and "Execute" loops based on "Knowledge", called MAPE-K [9]. Around this concept, we built the WOF [26] with design decisions mainly guided by reusability and genericity requirements: the framework should be maintainable and used in different application domains with different strategies (e.g., analysis approaches).

Seeking clarity, we have adopted some terms used for humans to refer to abilities a WO possesses. Awareness

2023, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

and wisdom both rely on knowledge. Inspired by [27], we give some definitions of those terms commonly used for humans [28] and present those we chose for WOs.

Data, it is a raw measurement, typed or not. The data are values obtained from a device or a software component. They represent a quantifiable observation. Generally, collected, stored, processed, transmitted and analyzed, the data have no meaning taken individually. To be relevant, they must be associated with a specific context. In this case, they become information [29].

Information The information provides the data with its context for analysis and processing. The information then becomes understandable by both humans and machines and gives a particular meaning to each data. Information is meaningful and allows us to better answer the questions When? Where? Who? What? etc. The answers to these simple questions do not allow us to make deductions. To do so, it is necessary to go up in abstraction towards knowledge [29]. i.e., placing your hand on a hot stove.

Knowledge: refers to information, inference rules and information deduced from them, for instance: "Turning on a heater will cause temperature change".

Awareness: represents the ability to collect - to provide internal data - on itself by itself. For instance, when an entity/object/device collects information and data about its capabilities *(what is intended to do)* and its use *(what it is asked to do)*. Capabilities are the services/functionalities the WO may render. They are implemented by methods that are invoked by the WO itself during the "dream" phase or from outside during the "awake" phase.

Wisdom: is the ability of WOs to analyze collected information and stored knowledge related to their capabilities and usage to output useful information. It is worth noticing that a WO is highly aware, while the converse is false (an aware object is not necessarily wise).

Semantic: is the subtle shades of meaning given to something so that it can be understood by humans as mentioned in [28]. This definition also applies to objects/devices, as semantic is used to communicate with humans. The value "100" of a variable "data" means nothing to an end user if we do not give her/him the information that it represents a percentage of humidity.

B. WO from an architectural view

From an architectural perspective, according to the target application, a WO may be considered as [9]:

- a stand-alone software entity (object, component, etc.),
- a software avatar designed to be a proxy for physical devices (e.g., a heater, vacuum cleaner, light bulb) [30],
- a software avatar designed to be a proxy for an existing software entity (object, component, etc.).

A WO is characterized by its:

- autonomy: it is able to operate with no human intervention,
- adaptability: it changes its behavior when its environment evolves,
- ability to communicate: with its environment according to a publish-subscribe paradigm.

In addition to the implementation of the MAPE-K loop, another concept, "phase transition", is used to separate the actions performed with a real-world impact (awake phase) from those without (dream phase). As illustrated in Figure 1, three different phases/super-states [8] of the WO [30][26] are defined, the "awake phase", the "dream phase" and the "idle phase".

During the awake phase, a WO responds to different services and requests, i.e., executes its methods called by other objects, monitors its execution and its usages (MAPE-K). Once the WO has finished, it switches to the idle phase. If a part of WO knowledge (e.g., a generated STG: Figure 2) requires analysis or if an AI plugin (see Figure 3) wants to analyze the WO itself, the WO can switch to dream phase. This phase is actually defined as a sub-phase of the idle phase, pushing the WO closer to the human, in other words, when it has nothing to do, it can dream.

Throughout this dream phase, a WO introspects its behavior and analyzes its knowledge without any effect on other objects of the system/real world. The ability of WOs to disconnect from the real world is a strong feature that distinguishes WSs from other self-adaptive systems such as multi-agent systems (MASs) [6]. Furthermore, within this phase, the WO switches from the MAPE-K feedback loop to IAPE-K feedback loop, this latter is nothing more than a technical adaptation of the MAPE-K feedback loop. Thus, rather than monitoring (M) its activity, the WO uses the "introspection" (I) mechanism to discover its behavior and any unusual operation of its use. We will not go into further detail as this topic is beyond the scope of this article.

C. Wise system & the framework

A WS is a set of WOs that communicate their states to the system as illustrated in Figure 3. It highlights a classical WS: a set of instantiated WOs (i.e., Application features, Managers, AIs at the system level) that contains the core (Figure 2) where the basic mechanisms for monitoring are defined. Each WO has three associated components:

- an event communication medium to publish its state to the system,
- a data Logger to log every interaction in/with the object,
- one or more AI plugins to provide the developer with the ability to add objects with different policies of introspection, monitoring, decision and action.

In addition, two kinds of WOs are defined in the system:

- manager: the WS may hold one or more managers (in Orange) that store the peering action/reaction among WOs, using for example Event-Condition-Action (ECA) rules,
- system AI: it manages the whole AI of the system, provided through WOs' AI plugins. AI denotes all activities needed for problem resolution, supervision, learning, analysis. We refer to those activities as Introspect-Monitor-Analyze-Decide-Act (IMADA-activities).

2023, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

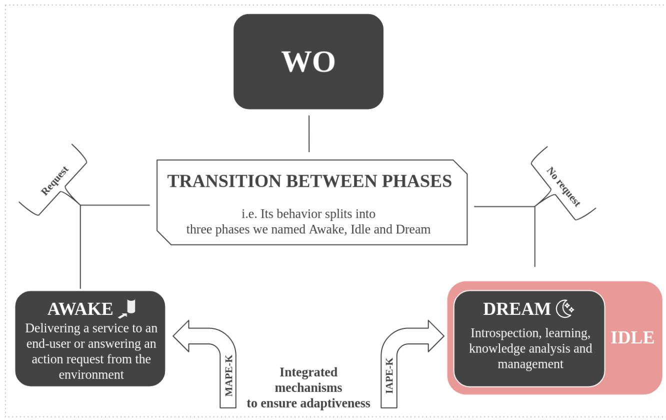

Figure 1. Three different phases/super-states of a WO.

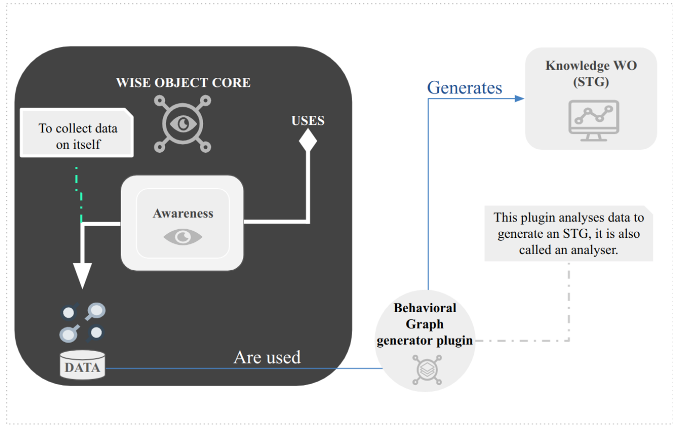

Figure 2. Generic functional architecture of a WO and its relation with an STG.

Regarding the core of the WO, it is designed to be connected to physical devices (e.g., heater, vacuum cleaner, light bulb) or logical entities [9]. It also contains an advanced mechanism to proxify each software entity to make it wise. In addition, the core contains the basic code expressing the two feedback loops (MAKE-k and IAPE-K) accessible to extend by experts depending on the domain/case in which the WO is used.

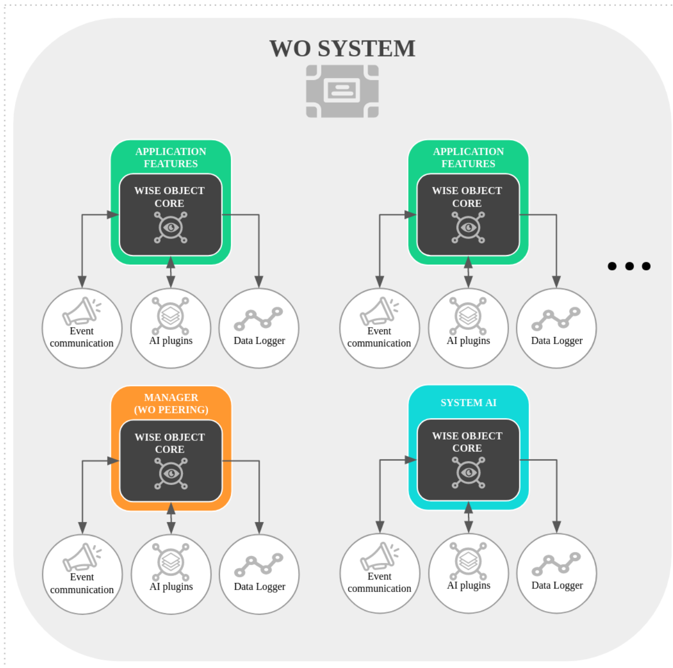

Figure 3. Global view of a WO system composed of a set of WOs, a manager and a system AI.

In order to build WSs, we developed in Java the WOF that provides all the WO mechanisms [31]. It provides the developers with “awareness”, “knowledge” and “wisdom” concepts, that are the core. From a system development perspective, the design behind WOF is driven by the following requirements:

- the separation of concerns between the business and AI features,
- minimum intrusion in the business code.

More specifically, the WOF separates the business logic layer from the AI layer, without change in the business layer. Thus, the designers and developers can focus on one hand on the business logic and on the other hand on the AI logic. Details about the WOF is beyond the scope of this article, for more details on the architecture used within the WOF, see [8].

D. The target problem

When designing a WS, developers provide a conceptual model describing, specifying the way they view the behavior of the system’s entities associated to WOs. Such models are represented using IOSTS and contain the semantic given by developers to WOs (Section III-B). The IOSTS formalism is mostly known in simplifying system modelling by allowing symbolic representation of parameters and variable values instead of concrete data values enumeration [32]. In addition, and as mentioned in Section II-B, the introspection mechanism helps the WO discover its behavior by using awareness to collect data about itself. These data will be used by the analyzers, for example by the STG generator plugin to build the STG that is a new WO knowledge. This new knowledge can be used by other analyzers like a graph matcher plugin that improves the STG through semantic, using the IOSTS given by an expert or a developer to the WO. This enhancement is based on the graph matching algorithm (Section IV), which will be the main focus of this paper.

This subsection is an implicit description of what is illustrated in Figure 4.

III. Behavioral Models, Definitions and Illustrations

Modeling the behavior of a system is enabled by tools and languages that result in informal, semi-formal (e.g., UML) or formal representations based on already proven theories [33] like graph theory. We have chosen STG and IOSTS graph-based theories to WO’s behavior representation, respectively

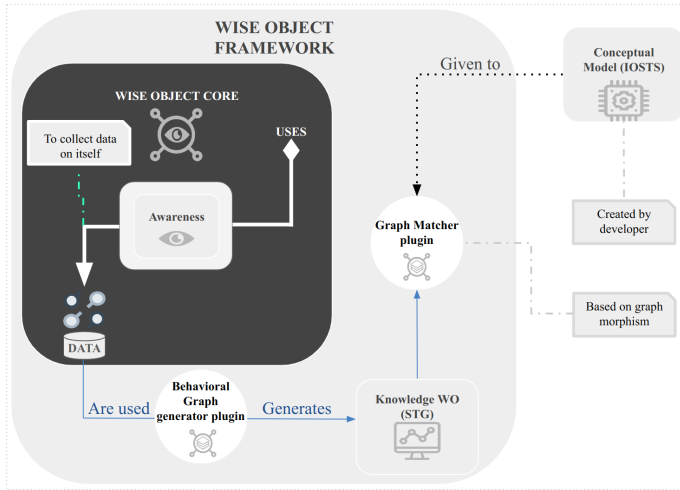

Figure 4. The matching algorithm and the WOF.

at the WO level (i.e., WO's generated view) and the conceptual level (i.e., developer's view).

A. Definition of an STG

An STG is a directed graph where vertices represent the states of an object and transitions represent the execution of its methods. Let us consider an object defined by its set of attributes A and its set of methods M. According to this information (A and M) on the object, the STG definition is given in Definition 1.

***Definition** 1:* An STG is defined by the triplet $G(V, E, L)$ where V and E are, respectively, the sets of vertices and edges, and L a set of labels.

- V is the set of vertices, with $|V| = n$ where each vertex represents a unique state of the object, and conversely, each state of the object is represented by a unique vertex. Therefore $v_i = v_j \Leftrightarrow i = j$ with $v_i, v_j \in V$ and $i, j \in [0, n[$.
- E is the set of directed edges where $\forall e \in E$, e is defined by the triplet $e = (v_i, v_j, m_k)$, such that $v_i, v_j \in V$ and $m_k \in M$. This triplet is called a transition labeled by m_k. The invocation of method m_k from state v_i switches the object to state v_j.
- L is a set of vertex labels where any label $l_i \in L$ is associated to v_i.
  A label l_i is the set of pairs $(att_j, value_{i,j})$ $\forall att_j \in A$, with $value_{i,j}$ the value of att_j in the state v_i and $Dom(att_j)$ the value domain of att_j, i.e., the set of $value_{i,j}$ for all i. By definition, 2 states v_i and v_j are different $v_i \neq v_j$, iff $\exists att_k \in A$, such that $value_{i,k} \neq value_{j,k}$. Conversely, if $\forall k \in [0, |A|[$ $value_{i,k} = value_{j,k}$, the states v_i and v_j are considered the same, i.e., $v_i = v_j$, thus $i = j$.

The matching algorithm we propose in Section IV takes as input an STG with a specific property we name exhaustiveness. The definition of "exhaustive STG" is given in Definition 2.

***Definition** 2:* An exhaustive STG is an STG such that from each vertex v_i there exist $|M|$ transitions, each labeled by a method m_k in M:

$$\forall v_i \in V, \forall m_k \in M, \exists v_j \in V | (v_i, v_j, m_k) \in E.$$

It is worth noting that v_i and v_j may be different or same states ($v_i \neq v_j$ or $v_i = v_j$).

Consequently, an exhaustive STG is deterministic, i.e., from any state, on any method invocation, the destination state is known. Moreover, the number of transitions $|E|$ in an exhaustive STG depends on the number of vertices $|V|$ and methods $|M|$ such that:

$$|V| \times |M| = |E|.$$

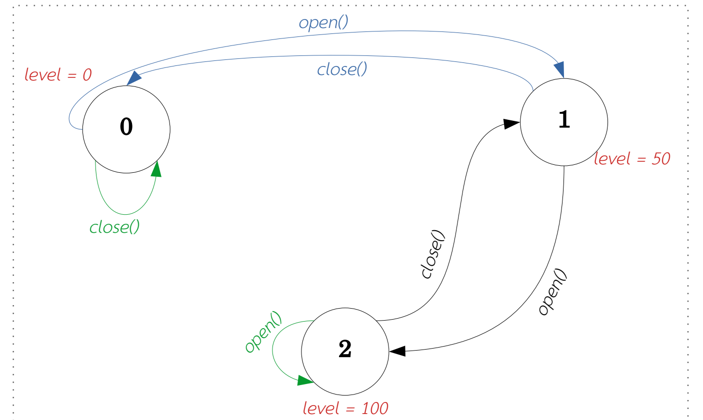

Figure 5. Example of an exhaustive STG.

Figure 5 illustrates an exhaustive STG for an object's behavior, defined by the attribute "level" ($A = \{level\}$) and 2 methods "open" and "close" ($M = \{open(), close()\}$). The methods "open" and "close" increase and decrease the level by 50, respectively. In the STG generated by an object for the shutter, except the methods that give semantic to the transitions, the states have no semantic. Considering the level is initialized to 0, the corresponding STG has 3 states and its exhaustive form has 6 transitions.

B. Definition of an IOSTS

An IOSTS is a directed graph whose vertices, called localities, represent different states of the system (in our case, the system is a software object) and whose edges are transitions. The localities are connected by transitions triggered by actions. In graph theory, an IOSTS allows us the definition of an infinite state transition system in a finite way, contrary to an STG where states are defined by discrete values. IOSTS are used to verify, test and control systems. Verification and testing are formal techniques for validating and comparing two views of a system while control is used to constrain the system behavior [20].

The definition of IOSTS given in Definition 3 is taken from [34][20] and especially from the use case given in [32].

***Definition** 3:* An IOSTS is a sixfold $\langle D, \Theta, Q, q_0, \Sigma, T\rangle$ such as:

- D is a finite set of typed data consisting of two disjoint sets of: variables X and action parameters P. The value domain of $d \in D$ is determined by $Dom(d)$.
- Θ : an initial condition expressed as a predicate on variables X.
- Q is a non-empty finite set of localities with $q_0 \in Q$ being the initial locality. A locality q is a set of states such that $q \in Dom(X)$, with $Dom(X)$ being the cartesian product of the domains of each $x \in X$:

$$Dom(X) = \prod_{\forall x \in X}^{X} Dom(x).$$

  A state is defined by a tuple of values for the whole variables.
- Σ is the alphabet, a finite, non-empty set of actions. It consists of the disjoint union of the set $\Sigma^?$ of input actions, the set $\Sigma^!$ of output actions, and the set $\Sigma^{\mathcal{T}}$ of internal actions. For each action a in Σ, its signature $sig(a) = \langle p_1, \ldots, p_k\rangle | p_i \in P$ is a tuple of parameters. The signature of internal actions is always an empty tuple.
- T is a finite set of transitions, such that each transition is a tuple $t = \langle q_o, a, G, A, q_d\rangle$ defined by:
  - a locality $q_o \in Q$, called the origin of the transition,
  - an action $a \in \Sigma$, called the action of the transition,
  - a boolean expression G on $X \cup sig(a)$ related to the variables and the parameters of the action, called the transition guard. Transition guards allow us to distinguish transitions that have the same origin and action but disjoint conditions to their triggering.
  - An assignment of the set of variables, of the form $(x := A^x)_{x \in X}$ such that for each $x \in X$, A^x is an expression on $X \cup sig(a)$. It defines the evolution of variable values during the transition,
  - a locality q_d, called the transition destination.

According to Definition 3, each variable has a subdomain in each locality. Thus, let us define the function $dom(q, x)$ that returns the definition domain of the variable $x \in X$ in the locality $q \in Q$; consequently $dom(q, x) \subseteq Dom(x)$.

Figure 6 shows an example of an IOSTS given by a developer to control a roller shutter. This IOSTS expresses that the roller shutter expects an input $up?/down? \in \Sigma^?$ carrying the parameter $step \in]0, 100]$, the relative elevation to respectively increase or decrease the shutter level. Let us note that the shutter elevation is between 0 and 100.

There are 2 localities:

- The locality where the system is closed (i.e., $height = 0$). If the system receives the $up?(step)$ command, the transition will be made from the $Closed$ to $Open$ locality by increasing the value of the $height$ variable by $step$, but if the system receives the $down?(step)$ action, it will not perform any operation (NOP).
- The locality where the system is open (i.e., $height \in]0, 100]$). If the system receives the action $up?(step)$, the transition will be reflexive from $Open$ to itself and will compute the value of the variable $height$ by executing this assignment $height = min(height + step, 100)$, the shutter elevation cannot be increased more than the maximum of elevation. If it receives the $down?(step)$ action and the action closes the shutter less than it is open ($step < height$), $height$ is decreased by $step$, otherwise the transition will be from the locality $Open$ to the locality $Close$ by assigning 0 to the variable $height$.

According to Definition 3, this IOSTS is composed of the sets of variables $X = \{height\}$ with $Dom(height) \in [0, 100]$ and parameters $P = \{step\}$ with $Dom(step) \in]0, 100]$, the set of localities $Q = \{Open, Closed\}$ and the set of actions $\Sigma = \{up?, down?\}$ where the signatures of the actions are $sig(up?) = sig(down?) = \langle step\rangle$. This IOSTS models an infinite state system based on 5 guarded transitions in T:

$$T = \langle \; t_{Close-Open}, \; t_{Open-Close}, \; t^1_{Open-Open}, \; t^2_{Open-Open}, \; t_{Close-Close} \; \rangle$$

such as:

$$\begin{aligned}
t_{Close-Open} &= \langle \; Open, up?(step), \\ &\quad True, height := height + step, \\ &\quad Open\rangle \\
t_{Open-Close} &= \langle \; Open, down?(step), \\ &\quad step \geq height, height := 0, \\ &\quad Close\rangle \\
t^1_{Open-Open} &= \langle \; Open, down?(step), \\ &\quad step < height, height := height \\ &\quad -step, Open\rangle, \\
t^2_{Open-Open} &= \langle \; Open, up?(step), \\ &\quad True, min(height + step, 100), \\ &\quad Open\rangle, \\
t_{Close-Close} &= \langle \; Close, down?(step), \\ &\quad True, NOP, \\ &\quad Close\rangle.
\end{aligned}$$

As can be noticed, there exists an infinity of paths and states represented by the variable $height$ since its domain is the interval $[0, 100]$.

IV. Graph matching algorithm

In this section, we introduce the matching algorithm we propose to relate WO's generated STG to developers' semantic expressed in an IOSTS. In the example of Figure 5, the generated STG is composed of states automatically labelled by the object: 0, 1 and 2 according to the value of attribute level: 0, 50, 100. The main challenge is how to match states 0, 1 and 2 to the localities defined by developers in the IOSTS of Figure 6.

A. Matching algorithm

Constraint: The STG and IOSTS must meet certain criteria to properly apply the algorithm.

2023, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

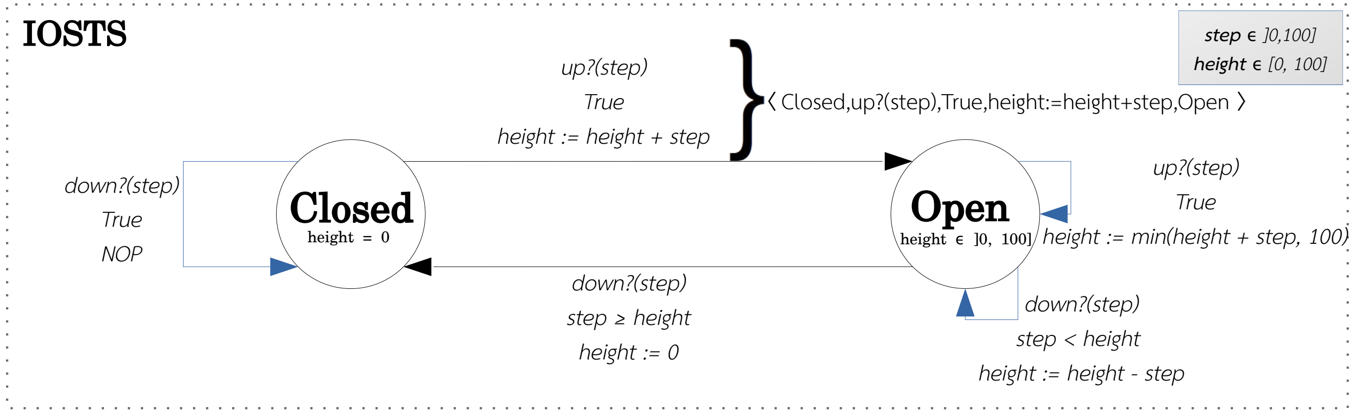

Figure 6. IOSTS representation of a roller shutter.

1) There are two equivalent characteristics: a variable x_e belonging to the set of variables X of the IOSTS and an attribute att_e belongs to the set of STG attributes A. Moreover, to simplify the problem in this paper, let us consider they are unique:
$$\exists! x_e \in X, \exists! att_e \in A | x_e \equiv att_e, \quad (1)$$
$x_e \equiv att_e$ means that both represent the same information, thus:
$$Dom(att_e) \subseteq Dom(x_e). \quad (2)$$
Let us note that $Dom(att_e)$ is a subset of $Dom(x_e)$ due to the fact that x_e is theoretically defined into the IOSTS and att_e is partially discovered by the WO at runtime.
2) The domains of x_e in the different localities in the IOSTS are disjoint:
$$\forall q, q' \in Q, q \neq q' \\ \Leftrightarrow \\ dom(q, x_e) \cap dom(q', x_e) = \emptyset$$

Algorithm: According to the definitions of STG and IOSTS, and both constraints, a vertex $v_i \in V$ matches a locality $q_j \in Q$ (noted $v_i \implies q_j$) if and only if $value_{i,e} \in dom(q_j, x_e)$, with $value_{i,e}$ the value of att_e in the vertex v_i:
$$\begin{gathered} \forall v_i \in V, \exists! q_j \in Q \\ value_{i,e} \in dom(q_j, x_e) \Leftrightarrow v_i \implies q_j. \end{gathered} \quad (3)$$

As the matching algorithm is a graph morphism, this latter needs to respect the structure of the matched graphs [35]. In our context, each vertex matches one locality and a locality is matched by at least one vertex. Moreover, the adjacency relations must be respected by the matching; if 2 vertices are linked by a transition in the STG, their matched localities are the same or linked by an equivalent transition in the IOSTS. The STG → IOSTS matching is a surjective $\mathcal{S}$ homomorphism, i.e., epimorphism [35] as illustrated in the formulas below:
$$\begin{aligned} \mathcal{S}\colon STG &\to IOSTS \\ V &\twoheadrightarrow \mathcal{S}(V) = Q, \end{aligned} \quad (4)$$
implies:
$$\mathcal{S}_E\colon E \to T, \mathcal{S}_E((v_i, v_j)) = (\mathcal{S}(v_i), \mathcal{S}(v_j)) \subset T. \quad (5)$$
For any transition $(v_i, v_j) \in E$ of STG, then $(\mathcal{S}(u_i), \mathcal{S}(v_j)) \in T$ is a transition of the IOSTS.

B. Stepwise matching algorithm

In this section, the matching algorithm will be presented step by step, as illustrated in Figure 7, to understand deeply how it works. The algorithm is divided into two steps, the first matches the attribute-variable according to their domain definition and defined by Function "*compatibleDomain*". The second step is devoted to compute the matching between states and localities, which implementation is described in Algorithm 1 (main algorithm). Furthermore, Algorithm 2 is the implementation details of the "*compatibleDomain*" function whose result will be part of the inputs to the main algorithm.

In detail, equations (1) and (2) are developed in the "*compatibleDomain*" function, which is exposed in algorithm 2. And Equation (3) is exposed in algorithm 1. This pseudo-code is the first version of the matching algorithm we have developed and tested as a first step towards human semantic.

C. First matching illustration

In the previous examples: the STG in Figure 5 is automatically generated by a WO and the IOSTS in Figure 6 is provided by a developer. Both represent the same roller shutter behavior. The STG uses discrete values with a level of opening of 50%, while the IOSTS uses continuous intervals, without any constraint on the step that is a real value.

Figure 8 illustrates the result of matching both graphs using our graph matcher implemented with Python. Localities in the IOSTS are *Closed* and *Open*, each containing variables with disjoint domains, in our example, a single variable named *height* that takes different values depending on its locality.

According to the constraints of the matching algorithm:

2023, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

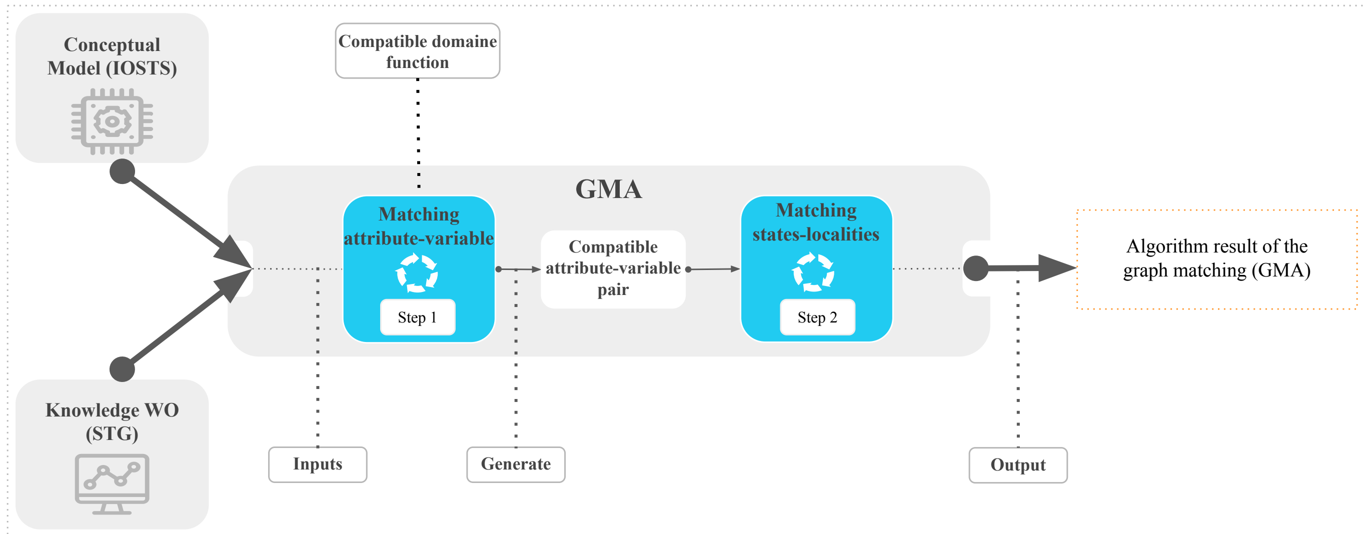

Figure 7. Illustration of stepwise matching algorithm applied on roller shutter (output in Figure 8).

1) there are equivalent characteristics between the STG and the IOSTS, the attribute "level" and the variable "height", respectively,
2) the domains of "height" in the different localities are disjoint from the others: in $Closed$ locality, the variable can only take the value 0 and in $Open$ locality, the variable can take any value in the range $]0, 100]$.

On the STG side, there are three vertices, each one labeled with a set of attribute-value pairs (att, value). In our case, the unique attribute $level$ takes the values $(0, 50, 100)$ respectively for (v_0, v_1, v_2). Therefore, to establish a correspondence between the two graphs, a comparison between the definition domain of the attribute $level$ in each vertex of the STG with the definition domain of the variable $height$ in each locality of the IOSTS must be done.

Those comparisons lead us to a correspondence of state v_0 with locality Closed meaning that the roller shutter is closed, and a correspondence of states v_1 and v_2 with locality Open meaning that the roller shutter is open.

D. Second matching illustration

The second illustration concerns a connected light bulb with RGB colours. To simplify the illustration, only two colors are considered and the off state of the light bulb is not considered. Figure 9 shows both simplified behavioural graphs of the light bulb: STG and IOSTS. The former is defined by the attribute "specter" ($A = \{specter\}$) and 2 methods "upFrequence" and "downFrequence" ($M = \{upFrequence(), downFrequence()\}$). The "upFrequence" and "downFrequency" methods increase and decrease the specter by 40nm, respectively. The second graph contains two localities $Green$ and $Blue$, each containing variables with disjoint domains, in our example, a single variable named $wavelength$ that takes different values depending on its locality.

The constraints of the matching algorithm are respected:

1) there are equivalent characteristics between the STG and the IOSTS, the attribute "specter" and the variable "wavelength", respectively,
2) the domains of "wavelength" in the different localities are disjoint from the others: in $Green$ locality, the variable can take any value in the range $]495, 570]$ and in $Blue$ locality, the variable can take any value in the range $[450, 495]$.

The algorithm lead us to a correspondence of states v_0, v_1 with locality $Blue$ meaning that the bulb is in blue color, and a correspondence of states v_2, v_3 with locality $Green$ meaning that the bulb is in green color.

V. Related Work

For many years, graphs have been used in several fields to represent complex problems in a descriptive way (e.g., maps, relationships between people profiles, public transportation, scene analysis, chemistry, molecular biology, and so on) for various purposes: analysis, operation, knowledge modeling, pattern detection, etc. Although initiated in the 18th century with Euler's work on the famous problem of Königsberg bridges [36], graph theory remains a powerful tool for software-intensive system development and an effective way of representing objects as proved in [37]. Since then, several approaches of graph matching have been developed and the first formulation of the graph matching problem was proposed by [38] and dates back to 1979. Several formulations appeared afterwards like convex-concave programming formulation, maximum common subgraph (MCS), the use of the Frobenius norm based on graph adjacency matrices to express the maximization or the minimization of non-overlapping edges between two graphs. In general, there exist two major formulations for the graph matching problem [39][40]:

2023, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

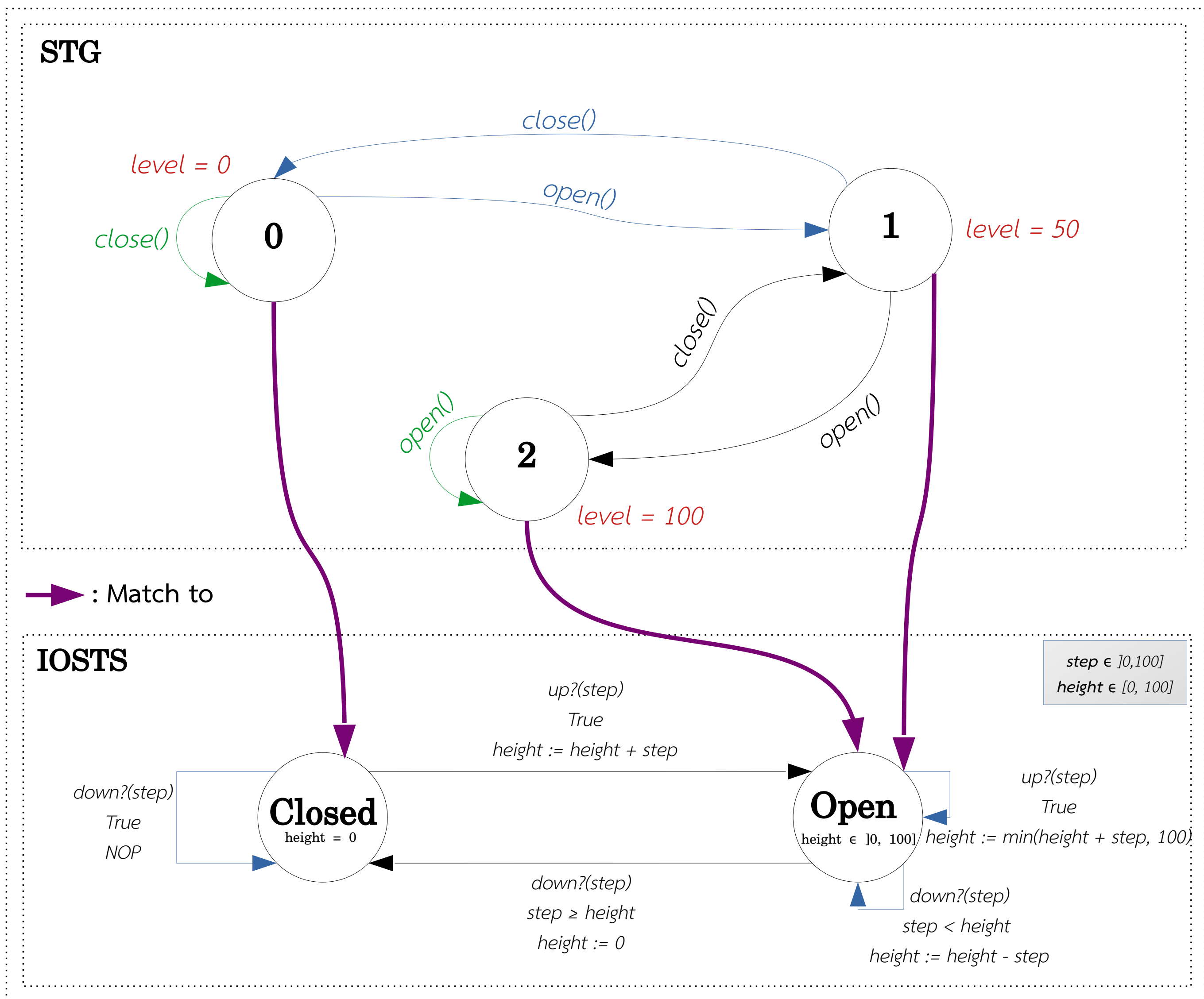

Figure 8. Algorithm result of the graph matching (roller shutter) [1].

- Exact Matching which is divided into two categories, (a) graph isomorphism, checks whether two graphs are the same. (b) subgraph isomorphism, checks whether the smallest graph is a subgraph of the biggest one. Both techniques are overly complex whether or not they check the one-to-one or many-to-one matching.
- Inexact Matching which is a term used in the case where it is impossible to find an isomorphism between two graphs. This form of matching is based on several approaches:
  - the maximum common subgraph, used in searching the similarity between graphs to know how different they are instead of a binary answer [41].
  - least-squares formulation, used in the case of weighted graphs to search for a matching that minimizes the total difference between all aligned edges through the use of the Frobenius norm for instance [41].
  - graph edit distance, used to find in a low cost the sequence of operations (i.e., deletion, insertion and substitution of vertices and edges) that transform one graph into another [42]. As this procedure is a hard combinatorial problem, another alternative called "beam search" is explained in details in [43].

In real applications, we often wish to match graphs of different sizes, which results in new techniques and norms as depicted in [39]. Moreover, as many formalisms have emerged so far, the correspondence between different representations of knowledge such as STGs and IOSTSs, has not been addressed yet at the best of our knowledge. Until now, the most well-known operation on graphs is the comparison of two or more graph representations that requires many theoretical

2023, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

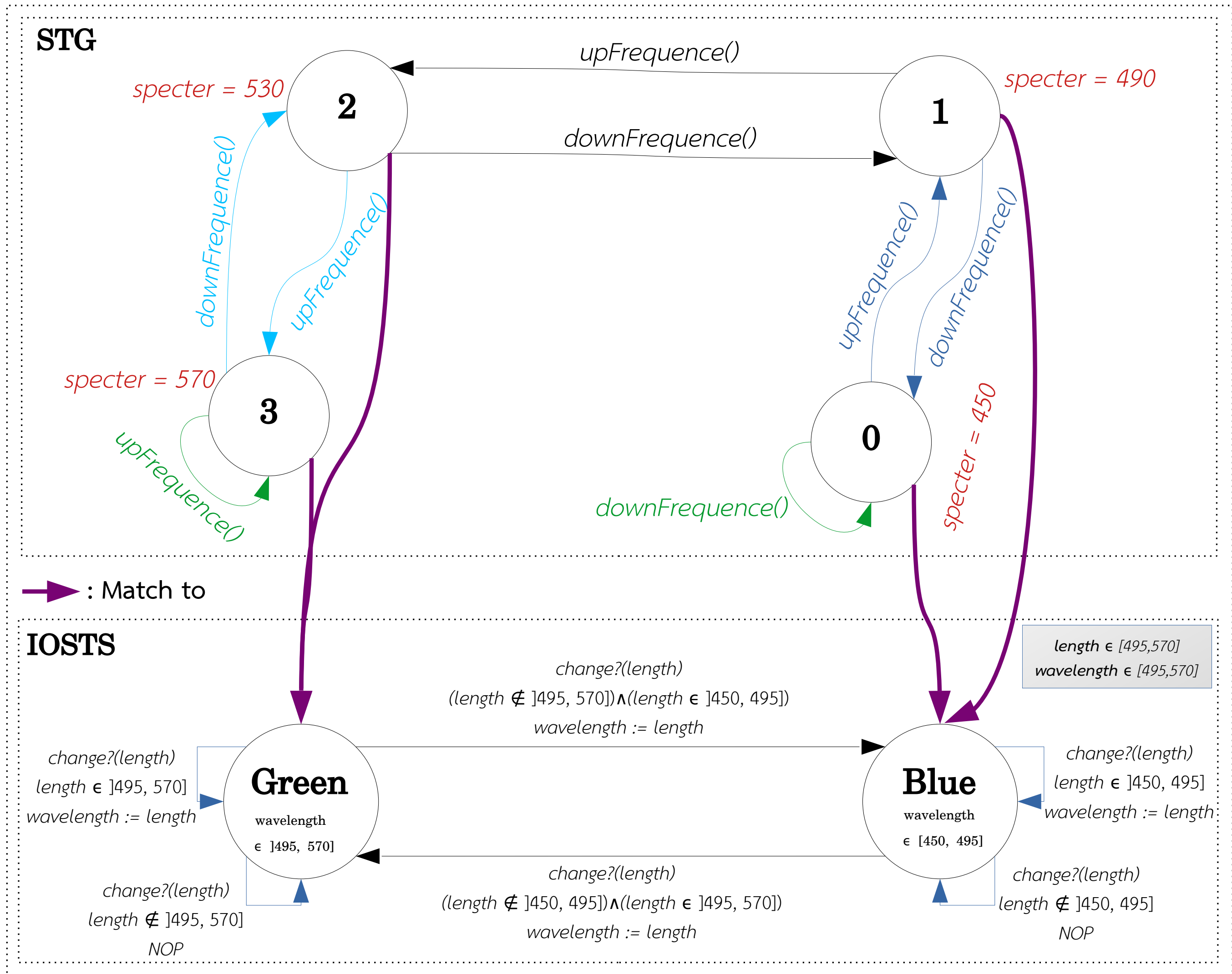

Figure 9. Algorithm result of graph matching (light bulb).

and complex concepts [21], like graph matching, a noisy version of graph isomorphism that is at the basis of our proposal in this paper. Finally, we mention that graph/sub-graph isomorphism is considered the most complex problem in graph matching as it has been proven to be NP-complete in [44][45][46] Moreover, for certain types of graphs under given constraints, the complexity of the isomorphism has been proven of polynomial type with a huge cost [47].

Using the constraints presented in Section IV to match two knowledge representations (STG - IOSTS) lead us to the exact matching problem. To understand the situation, we need to consider the matching from both perspectives: software (i.e., numerical and structural) and human (i.e., semantic). According to the software, and since the matching preserves the structure and the transitions between both formalisms, the matching is always exact between “states” and “localities”, which gives an epimorphism (Equations (4) and (5)). From a human perspective, we will always have an exact matching based on semantic as illustrated in Figures 8 and 9. The question is how to match STGs and IOSTs when constraints are expanded to include more than one equivalent attribute-variable. In this case, we should adopt an inexact matching approach so that the algorithm generates more than one result (see future work for more details).

VI. Conclusion and Future Work

In this paper, we addressed the problem of relating numerical representations generated by WSs to developers’ semantic. The contribution is a matching algorithm that computes a morphism between two behavioral graphs:

1) an STG generated by a WO along its learning process,
2) an IOSTS representing a developer conceptual view.

The algorithm extends a WO’s view with semantic that allows it to communicate with humans. From the developer’s perspective, the resulted matching may help him/her discover errors and/or inconsistencies between the conceptual view and

2023, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

Algorithm 1 Main algorithm: Graph matching algorithm

1: **Inputs:**
$iosts$: IOSTS,
stg: Exhaustive STG
2: **Outputs:**
$match$: Dictionary<state, locality>
3: **Locales:**
Set of possible attribute/variable pairs
E: Set Of Tuples< (attribute, variable) >
Possible matches for each possible equivalent pair
M: Dictionary< (attribute, variable), <state, locality>>
4: **Initialize:**
Build possible equivalent attribute/variable pairs, such that $dom(a) \subseteq dom(x)$ (Algorithm 2)
$E \leftarrow compatibleDomain(stg.A, iosts.X)$
5: **for** $(a, x) \in E$ **do**
6: **for** $v_i \in stg.V$ **do**
7: # Get the locality where the domain of variable x contains the value of a in v_i, according to the second constraint, q_i is unique
8: $q_i \leftarrow iosts.getLocality(x, v_i.getValue(a))$
9: $M((a, x))(v_i) \leftarrow q_i$
10: **end for**
11: # As the matching is a surjective application, remove the pair if it does not generate surjective matching
12: **if** $M((a, x)).getKeys() \neq stg.V$
13: **or** $M((a, x)).getValuesAsSet() \neq iosts.Q$ **then**
14: E.remove((a,x))
15: M.remove((a,x))
16: **else**
17: # If the application is surjective (Equation (4)), check the transitions' consistency (Equation (5))
18: **for** $e \in stg.E$ **do**
19: $v_1 \leftarrow e.getSource()$
20: $v_2 \leftarrow e.getDestination()$
21: $q_1 = M((a, x))(v_1)$
22: $q_2 = M((a, x))(v_2)$
23: **if** $iosts.getTransition(q_1, q_2)$ **is null then**
24: E.remove((a,x))
25: M.remove((a,x))
26: **end if**
27: **end for**
28: **end if**
29: **end for**
30: # Checking that just only one matching exists according to constraints defined in Section IV-A
31: **if** $M.getKeys().size() == 1$ **then**
32: $match \leftarrow M.getValues()[1]$
33: **else**
34: $exception($"Required conditions not satisfied"$)$
35: **end if**
36: **return** $match$

Algorithm 2 Variable matching algorithm ("compatibleDomain" function)

1: **Inputs:**
$iosts$: IOSTS,
stg: Exhaustive STG
2: **Outputs:**
Set of possible attribute/variable pairs such that $Dom(attribute) \subseteq Dom(variable)$
E: Set Of Tuples< (attribute, variable) >
3: # Build all possible equivalent attributes-variables using the cartesian product between A and X
4: **for** $cartesian \in product(stg.A, iosts.X)$ **do**
5: # Keep attribute-variable pairs, such that $Dom(a) \subseteq Dom(x)$
6: **if** $Dom(cartesian.getAttribute()) \subseteq Dom(cartesian.getVariable())$ **then**
7: $E.add(cartesian)$
8: **end if**
9: **end for**
10: **return** E

the system implementation. In its first version, the algorithm has obviously several limitations, the strongest being over the number of equivalent attributes/variables in STG/IOSTS. Another limitation is the constraint on the existence of only one matching between an STG and an IOSTS, without omitting the problem of inexact matching. Ongoing work is being done to gradually generalize the algorithm and raise those restrictions. The graph matching algorithm being an NP-complete problem, we envisage the use of ontologies in a matrix form through two main matrices, termed "Semantic Matrix" and "Graph Matching Matrix". Moreover, we initiated a France-Canada innovation project to apply our approach to help create assistive scenarios [10][11] for elderly people, in the context of smart home.

ACKNOWLEDGMENT

This research was supported by French National Research Agency (ANR), AI Ph.D funding project.

REFERENCES

[1] A. Dahhani, I. Alloui, S. Monnet, and F. Vernier, "Towards a semantic model for wise systems: A graph matching algorithm," Proceedings of the Sixteenth International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP 2022), IARIA, 2023, pp. 27–34.

[2] E. Nascimento, A. Nguyen-Duc, I. Sundbø, and T. Conte, "Software engineering for artificial intelligence and machine learning software: A systematic literature review," 2020.

[3] R. Feldt, F. G. D. O. Neto, and R. Torkar, "Ways of applying artificial intelligence in software engineering," 2018 IEEE/ACM 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), pp. 35–41, 2018.

[4] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in International Conference on Software Engineering (ICSE 2019) - Software Engineering in Practice track, May 2019. ICSE 2019 Best Paper Award.

2023, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

[5] E. Kusmenko, S. Pavlitskaya, B. Rumpe, and S. Stüber, "On the engineering of ai-powered systems," in 2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW), pp. 126–133, 2019.
[6] R. A. Flores-Mendez, "Towards a standardization of multi-agent system framework," XRDS, vol. 5, pp. 18–24, jun 1999.
[7] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-agent systems: A survey," IEEE Access, vol. 6, pp. 28573–28593, 2018.
[8] S. Lejamble., I. Alloui., S. Monnet., and F. Vernier., "A new software architecture for the wise object framework: Multidimensional separation of concerns," in Proceedings of the 17th International Conference on Software Technologies - ICSOFT, pp. 567–574, INSTICC, SciTePress, 2022.
[9] I. Alloui and F. Vernier, "WOF: Towards Behavior Analysis and Representation of Emotions in Adaptive Systems," Communications in Computer and Information Science, vol. 868, pp. 244–267, 2018.
[10] D. Bonino and F. Corno, "Dogont - ontology modeling for intelligent domotic environments," in The Semantic Web - ISWC 2008, pp. 790–803, Springer Berlin Heidelberg, 2008.
[11] H. Kenfack Ngankam, H. Pigot, M. Frappier, C. H. Souza Oliveira, and S. Giroux, "Formal specification for ambient assisted living scenarios," UCAmI, pp. 508–519, 2017.
[12] J.-B. Woo and Y.-K. Lim, "User experience in do-it-yourself-style smart homes," in Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing, pp. 779–790, 2015.
[13] R. Radziszewski, H. Ngankam, H. Pigot, V. Grégoire, D. Lorrain, and S. Giroux, "An ambient assisted living nighttime wandering system for elderly," in Proceedings of the 18th International Conference on Information Integration and Web-Based Applications and Services, iiWAS '16, pp. 368–374, Association for Computing Machinery, 2016.
[14] R. S. Michalski, "A theory and methodology of inductive learning," in Machine Learning: An Artificial Intelligence Approach, pp. 83–134, Springer Berlin Heidelberg, 1983.
[15] M. Weiser, "The computer for the 21st century," Scientific American, vol. 265, no. 3, pp. 66–75, 1991.
[16] M. Weiser and J. S. Brown, "Designing calm technology," PowerGrid Journal, vol. 1, pp. 75–85, 1996.
[17] A. Tugui, "Calm technologies in a multimedia world," Ubiquity, vol. 2004, pp. 1–5, 2004.
[18] I. Alloui and F. Vernier, "A Wise Object Framework for Distributed Intelligent Adaptive Systems," in ICSOFT 2017, the 12th International Conference on Software Technologies, pp. 95–104, 2017.
[19] C. Constant, T. Jéron, H. Marchand, and V. Rusu, "Validation of Reactive Systems," in Modeling and Verification of Real-TIME Systems - Formalisms and software Tools, pp. 51–76, Hermès Science, 2008.
[20] C. Constant, T. Jéron, H. Marchand, and V. Rusu, "Integrating Formal Verification and Conformance Testing for Reactive Systems," IEEE Transactions on Software Engineering, vol. 33, no. 8, pp. 558–574, 2007.
[21] M. R. Garey and D. S. Johnson, "Computers and intractability. a guide to the theory of np-completeness.," Journal of Symbolic Logic, vol. 48, no. 2, pp. 498–500, 1983.
[22] V. A. Cicirello, "Survey of graph matching algorithms," technical report, Geometric and Intelligent Computing Laboratory, Drexel University, 1999.
[23] B. Kitchenham, "A methodology for evaluating software engineering methods and tools," in Experimental Software Engineering Issues: Critical Assessment and Future Directions (H. D. Rombach, V. R. Basili, and R. W. Selby, eds.), (Berlin, Heidelberg), pp. 121–124, Springer Berlin Heidelberg, 1993.
[24] B. W. Boehm, "Software engineering - as it is," IEEE Trans. Computers, vol. 25, no. 12, pp. 1226–1241, 1976.
[25] D. Torre, M. Genero, Y. Labiche, and M. Elaasar, "How consistency is handled in model-driven software engineering and UML: an expert opinion survey," Software Quality Journal, pp. 1–53, Apr. 2022.
[26] I. Alloui, D. Esale, and F. Vernier, "Wise objects for calm technology," in Proceedings of the 10th International Conference on Software Engineering and Applications - ICSOFT-EA, (ICSOFT 2015), pp. 468–471, INSTICC, SciTePress, 2015.
[27] T. Davenport and L. Prusak, Working Knowledge: How Organizations Manage What They Know, vol. 1. Harvard Business School Press, 1998.
[28] "Cambridge Dictionary Online," 2022.
[29] H. K. Ngankam, Modèle Sémantique d'Intelligence Ambiante pour le Développement Do-It-Yourself d'Habitats Intelligents. Theses, Faculté des sciences, université de sherbrooke, 2019.
[30] I. Alloui, E. Benoit, S. Perrin, and F. Vernier, "Wise objects for IoT (WIoT): Software framework and experimentation," Communications in Computer and Information Science, pp. 349–371, 2019.
[31] I. Alloui, E. Benoit, S. Perrin, and F. Vernier, "Wiot: Interconnection between wise objects and iot," in ICSOFT 2018, the 13th International Conference on Software Technologies, 2018.
[32] P. Moreaux, F. Sartor, and F. Vernier, "An effective approach for home services management," in 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing, pp. 47–51, 2012.
[33] M. N. Nicolescu and M. J. Matarić, "Extending behavior-based systems capabilities using an abstract behavior representation," in AAAI 2000, pp. 27–34, 2000.
[34] V. Rusu, H. Marchand, and T. Jéron, "Automatic verification and conformance testing for validating safety properties of reactive systems," in Formal Methods 2005 (FM05), vol. 3582 of Lecture Notes in Computer Science, pp. 189–204, Springer-Verlag, 2005.
[35] G. Hahn and C. Tardif, "Graph homomorphisms: structure and symmetry," in Graph Symmetry: Algebraic Methods and Applications, pp. 107–166, Springer Netherlands, 1997.
[36] H. Sachs, M. Stiebitz, and R. Wilson, "An historical note: Euler's königsberg letters," Journal of Graph Theory, vol. 12, pp. 133 – 139, 2006.
[37] M. A. Eshera and K.-S. Fu, "An image understanding system using attributed symbolic representation and inexact graph-matching," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, pp. 604–618, 1986.
[38] W.-H. Tsai and K.-S. Fu, "Error-correcting isomorphisms of attributed relational graphs for pattern analysis," IEEE Transactions on Systems, Man, and Cybernetics, vol. 9, no. 12, pp. 757–768, 1979.
[39] M. Zaslavskiy, L'alignement de graphes : applications en bioinformatique et vision par ordinateur. Theses, École Nationale Supérieure des Mines de Paris, Jan. 2010.
[40] E. Bengoetxea, Inexact Graph Matching Using Estimation of Distribution Algorithms. PhD thesis, Ecole Nationale Supérieure des Télécommunications, Paris, France, Dec 2002.
[41] J. R. Ullmann, "An algorithm for subgraph isomorphism," J. ACM, vol. 23, pp. 31–42, jan 1976.
[42] H. Bunke and G. Allermann, "Inexact graph matching for structural pattern recognition," Pattern Recognition Letters, vol. 1, no. 4, pp. 245–253, 1983.
[43] M. Neuhaus, K. Riesen, and H. Bunke, "Fast suboptimal algorithms for the computation of graph edit distance," in Structural, Syntactic, and Statistical Pattern Recognition, (Berlin, Heidelberg), pp. 163–172, Springer Berlin Heidelberg, 2006.
[44] D. A. Basin, "A term equality problem equivalent to graph isomorphism," Information Processing Letters, vol. 51, no. 2, pp. 61–66, 1994.
[45] M. R. Garey and D. S. Johnson, Computers and intractability: A guide to the theory of NP - completeness. W.H. Freeman and Co., 1979.
[46] M. A. Abdulrahim, Parallel algorithms for labeled graph matching. Colorado School of Mines1500 Illinois St. Golden, CO, 1998.
[47] J. E. Hopcroft and J. K. Wong, "Linear time algorithm for isomorphism of planar graphs (preliminary report)," in Proceedings of the Sixth Annual ACM Symposium on Theory of Computing, STOC '74, (New York, NY, USA), pp. 172–184, Association for Computing Machinery, 1974.

2023, © Copyright by authors, Published under agreement with IARIA - www.iaria.org