Towards a Semantic Model for Wise Systems A Graph Matching Algorithm

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 named Wise Objects, able to autonomously learn how they behave and how they are used. They 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. Construction of such systems requires at least two views: on one hand, a conceptual view relying on knowledge given by developers to either control or specify the expected system behavior. On the other hand, wise systems are provided with mechanisms to generate their own view based on behavior-related knowledge acquired during their learning processes. The problem is that, while a conceptual view is understandable by humans (i.e., developers, end users, 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. This provides the ability to make wise systems' generated knowledge understandable by humans and to enable human evaluation of wise systems' outputs.**

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

I. INTRODUCTION

Software systems are now everywhere in our daily life. Their usage may vary depending on the end user and may evolve in time. A wise system should be able to adapt itself gracefully according to its usage. It can be seen as a particular Multi-Agent System [1][2] that monitors only its internal changes and does not observe its external environment. To tackle this issue, we have previously proposed Wise Objects (WO) and Wise systems. 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). A Wise system is simply a collection of communicating WOs. The main specificity of a WO is that it is able to autonomously learn about itself: 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 provided by a WO. Then, the collected monitoring data can feed a learning process to be able to determine usual and unusual behavior (for instance). The autonomic behavior is a key property: the end user should not be required to interact with the WO to help it in its learning process. An example is that of a home-automation system that collects someone's behavior in a room and analyses 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 [3] (WOF) to help developers design, deploy and evolve wise systems. Generally, knowledge in Artificial Intelligent (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 [4][5]. In [4], functional behavior as well as inter-operation of system entities are described a priori using state-diagrams. Reference [5] goes a step forward by combining ontologies to design ambiant 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 [6][7]. *In the second case*, knowledge is provided by the AI-enabled system in

 ISBN: 978-1-61208-990-4

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 [8].

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 [9] (Calm technology). Mark Weiser and John Seely Brown in [10] describe calm technology as "that which informs but does not demand our(users) focus or attention". The WO also analyses 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) [11]. The downside of an STG generated by a WO is that numeric data provided has no meaning for humans. In the literature [12][13], others graphs like Input Output Symbolic Transition System (IOSTS) are often used to model the behavior of systems to manage them using oracle or controller synthesis. Since this type of graph is conceptually understandable by humans and it has semantics, it can increase the knowledge of WO and brings semantic to STGs.

Our proposal is to extend the generated knowledge with the conceptual knowledge using a matching algorithm based on graph morphism [14][15]. This provides the ability to make wise systems' generated knowledge understandable by humans and to enable human evaluation of wise systems' 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 wise system's learning process. In this way, we use two state-based formalisms:

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

For many years, graphs have been used in many fields to represent complex problems in a descriptive way (e.g., maps, relationships between people profiles, public transportation) for various purposes: analysis, operation, knowledge modeling, etc. Although initiated in the 18th century with Euler's work on the now famous problem of Königsberg bridges [16], Graph theory remains a powerful tool for software-intensive system development. Among the most well-known operations on graphs is the comparison of two or more graph representations that requires many theoretical and complex concepts [14], like graph matching and graph morphism. These are at the basis of our proposal in this work. The rest of the paper is organised as follows. Section II 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, before Section V, which concludes the paper.

II. Basic idea & architectural view

The first step toward WO concept is to respect the notion of "calm technolog" claimed by Mark Weiser and John Seely Brown in [17], by giving an entity the ability to autonomously adapt itself to its usage. We sketch in this section an overall view of how we designed the WO concept to meet this requirement.

A. Basic idea & definitions

The basic idea underlying the WO concept is to give a software entity (object, component, subsystem) 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 [3]. At the core of this concept, we built the WOF [18] 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 borrowed some terms used for humans to refer to abilities a WO possesses. Awareness and wisdom both rely on knowledge. Inspired by [19], we give some definitions of those terms commonly used for humans [20] and present those we chose for WOs.

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, it is 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 to analyse 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.

Semantic: is the meaning given to something so that it can be understood by humans as mentioned in [20]. 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 [3]:

- a stand-alone software entity (object, component, etc.),

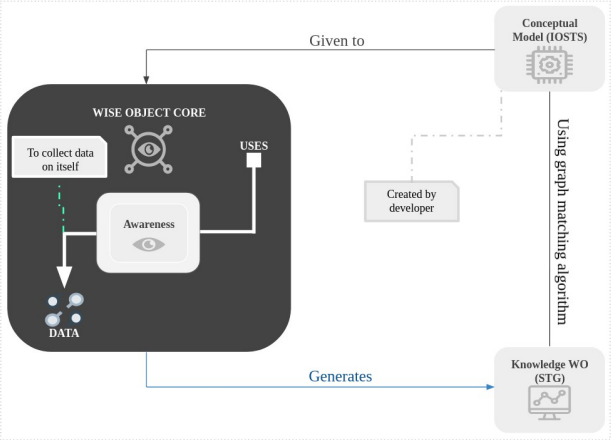

Figure 1. Generic functional architecture of a WO.

- a software avatar designed to be a proxy for physical devices (e.g., a heater, vacuum cleaner, light bulb) [21],
- a software avatar designed to be a proxy for an existing software entity (object, component, etc.).

A WO is characterised 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.

Figure 1 illustrates a partial view of a WO's functional architecture defined in the WOF. As depicted, the WO uses awareness to collect data on itself, either in simulation mode or usage mode. It analyses those data and generates a behavioral graph represented by an STG (Section III-A). States are constructed by the WO from attribute values of invoked methods and transitions from method invocations. On another hand, when designing an application, 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 [22].

The STG and IOSTS will be given to the matching algorithm (Section IV) by the WO to automatically add the developer's semantic to the STG. A concrete implemented example will illustrate this matching.

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 [23] like graph theory. We have chosen STG and IOSTS graph-based theories to WO's behavior representation, respectively at the conceptual level (i.e., developer's view) and the wise system level (WO's generated 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 $v_i \neq v_j$, $iff \quad \exists att_k \in A$, 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 2 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

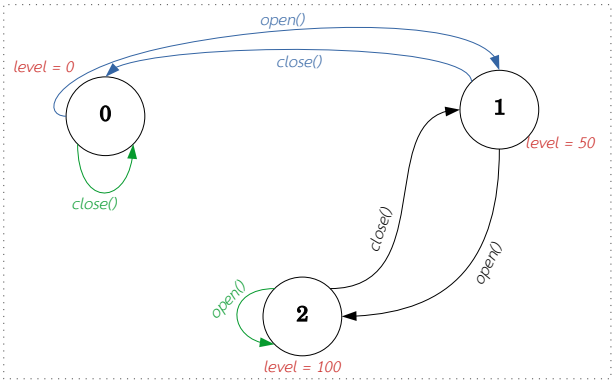

Figure 2. Example of an exhaustive STG.

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 [13].

The definition of IOSTS given in Definition 3 is taken from [13][24] and especially from the use case given in [22].

***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 $x \in X$. Let us note that a state is defined by a single 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 this definition, 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 3 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:

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

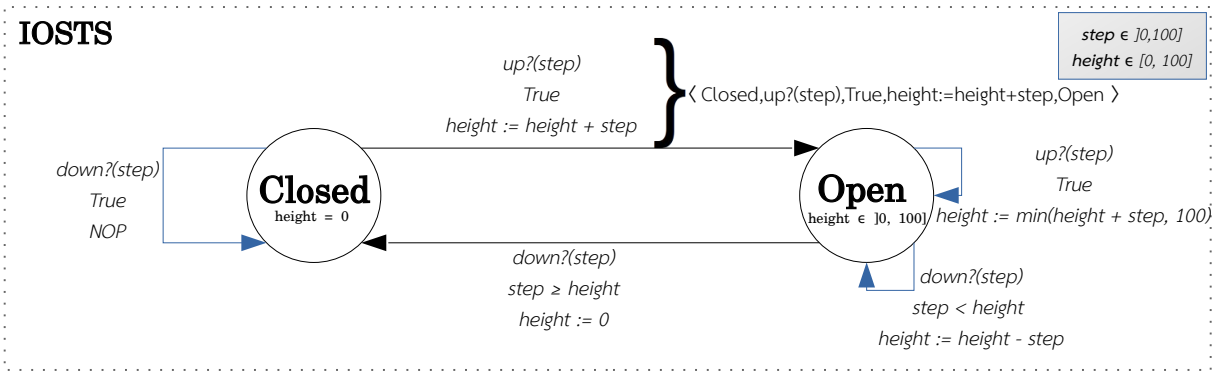

Figure 3. IOSTS representation of a roller shutter.

such as:

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

As can be noticed, there exists an infinity of paths and states represented by the variables $height$ 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 2, 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 3.

A. Matching algorithm

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

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:

$$\exists! x_e \in X, \exists! att_e \in A | x_e \equiv att_e,$$

$x_e \equiv att_e$ means that both represent the same information, thus:

$$Dom(att_e) \subseteq Dom(x_e).$$

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:

$$\forall q, q' \in Q, dom(q, x_e) \cap dom(q', x_e) = \emptyset,$$

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:

$$\forall v_i \in V, \exists! q_j \in Q$$
$$value_{i,e} \in dom(q_j, x_e) \Leftrightarrow v_i \implies q_j.$$

As the matching algorithm is a graph morphism, this latter needs to respect the structure of the matched graphs [25]. 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 $\rightarrow$ IOSTS maching is surjective homomorphism called epimorphism [25].

The pseudo-code in Algorithm 1 is the first version of the matching algorithm we have developed.

B. Matching illustration

In the previous examples: the STG in Figure 2 is automatically generated by a WO and the IOSTS in Figure 3 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 4 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

Algorithm 1 Graph matching algorithm

Inputs:
  $iosts$: IOSTS,
  stg: Exhaustive STG
Outputs:
  $match$: Dictionary<state, locality>
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>>
Initialize:
  # Build possible equivalent attribute/variable pairs, such that $dom(a) \subseteq dom(x)$
  $E \leftarrow compatibleDomain(stg.A, iosts.X)$
for $(a, x) \in E$ **do**
  for $v_i \in stg.V$ **do**
# 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
    $q_i \leftarrow iosts.getLocality(x, v_i.getValue(a))$
    $M((a, x))(v_i) \leftarrow q_i$
  end for
# As the matching is a surjective application, remove the pair if it does not generate surjective matching
  if $M((a, x)).getKeys() \neq stg.V$
  or $M((a, x)).getValuesAsSet() \neq iosts.Q$ **then**
    E.remove((a,x))
    M.remove((a,x))
  else
# If the application is surjective, check the transitions' consistency
    for $e \in stg.E$ **do**
      $v_1 \leftarrow e.getSource()$
      $v_2 \leftarrow e.getDestination()$
      $q_1 = M((a, x))(v_1)$
      $q_2 = M((a, x))(v_2)$
      if $iosts.getTransition(q_1, q_2)$ **is null then**
        E.remove((a,x))
        M.remove((a,x))
      end if
    end for
  end if
end for
# Checking that just only one matching exists according to constraints defined in Section IV-A
if $M.getKeys().size() == 1$ **then**
  $match \leftarrow M.getValues()[1]$
else
  $exception$("Required conditions not satisfied")
end if
return $match$

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:

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_1 with locality Closed meaning that the roller shutter is closed, and a correspondence of states v_2 and v_3 with locality Open meaning that the roller shutter is open.

This first version of the matching algorithm has been developed and tested as a first step towards human semantics.

V. Conclusion and Future Work

In this paper, we addressed the problem of relating numerical representations generated by wise systems to developers' semantics. 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.

That algorithm extends a WO's view with semantics that allow it to communicate with humans. From the developer's perspective, the resulted matching may help developers discover errors and/or inconsistencies between the conceptual view and 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. Ongoing work is being done to gradually generalize the algorithm and raise those restrictions. We also intend to investigate other matching algorithms towards other semantic formalisms than IOSTS, such as ontology and scenario-based ones [4][5].

Acknowledgment

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

References

[1] R. A. Flores-Mendez, "Towards a standardization of multi-agent system framework," XRDS, vol. 5, no. 4, pp. 18–24, 1999.
[2] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-agent systems: A survey," IEEE Access, vol. 6, pp. 28573–28593, 2018.

 ISBN: 978-1-61208-990-4

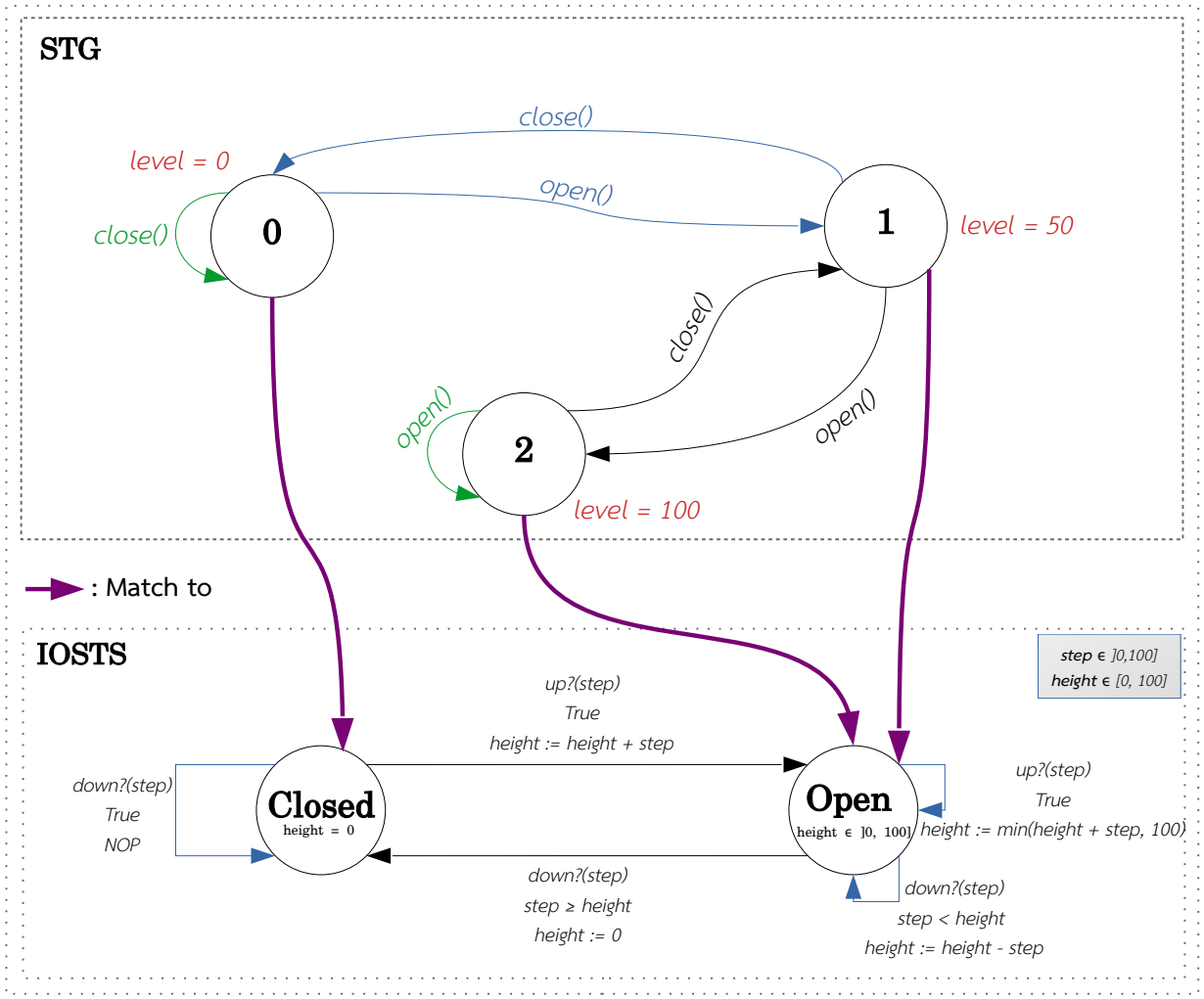

Figure 4. Algorithm result of the graph matching.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] R. S. Michalski, "A theory and methodology of inductive learning," in Machine Learning: An Artificial Intelligence Approach, pp. 83–134, Springer Berlin Heidelberg, 1983.

[9] M. Weiser, "The computer for the 21st century," Scientific American, vol. 265, no. 3, pp. 66–75, 1991.

[10] A. Tugui, "Calm technologies in a multimedia world," Ubiquity, vol. 2004, pp. 1–5, 2004.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] V. A. Cicirello, "Survey of graph matching algorithms," technical report, Geometric and Intelligent Computing Laboratory, Drexel University, 1999.

[16] H. Sachs, M. Stiebitz, and R. Wilson, "An historical note: Euler's königsberg letters," Journal of Graph Theory, vol. 12, pp. 133 – 139, 2006.

[17] M. Weiser and J. S. Brown, "Designing calm technology," PowerGrid Journal, vol. 1, pp. 75–85, 1996.

[18] 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.

[19] T. Davenport and L. Prusak, Working Knowledge: How Organizations Manage What They Know, vol. 1. Harvard Business School Press, 1998.

[20] "Cambridge Dictionary Online," 2022.

[21] I. Alloui, E. Benoit, S. Perrin, and F. Vernier, "Wise objects for IoT

 ISBN: 978-1-61208-990-4

(WIoT): Software framework and experimentation," Communications in Computer and Information Science, pp. 349–371, 2019.

[22] 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.

[23] M. N. Nicolescu and M. J. Matarić, "Extending behavior-based systems capabilities using an abstract behavior representation," in AAAI 2000, pp. 27–34, 2000.

[24] 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.

[25] G. Hahn and C. Tardif, "Graph homomorphisms: structure and symmetry," in Graph Symmetry: Algebraic Methods and Applications, pp. 107–166, Springer Netherlands, 1997.

 ISBN: 978-1-61208-990-4