

<sub>1</sub> A Unified Modeling Framework to Abstract  
<sub>2</sub> Knowledge of Dynamically Adaptive Systems

<sub>3</sub> Ludovic Mouline

<sub>4</sub> April 15, 2019



---

# Abstract

---

**Vision:** As state-of-the-art techniques fail to model efficiently the evolution and the uncertainty existing in dynamically adaptive systems, the adaptation process makes suboptimal decisions. To tackle this challenge, modern modeling frameworks should efficiently encapsulate time and uncertainty as first-class concepts.

*Context* Smart grid approach introduces information and communication technologies into traditional power grid to cope with new challenges of electricity distribution. Among them, one challenge is the resiliency of the grid: how to automatically recover from any incident such as overload? These systems therefore need a deep understanding of the ongoing situation which enables reasoning tasks for healing operations. **Abstraction** is a key technique that provided an illuminating description of systems, their behaviors, and/or their environments alleviating their complexity. **Adaptation** is a cornerstone feature that enables reconfiguration at runtime for optimizing software to the current and/or future situation.

Abstraction technique is pushed to its paramountcy by the model-driven engineering (MDE) methodology. However, information concerning the grid, such as loads, is not always known with absolute confidence. Through the thesis, this lack of confidence about data is referred to as **data uncertainty**. They are approximated from the measured consumption and the grid topology. This topology is inferred from fuse states, which are set by technicians after their services on the grid. As humans are not error-free, the topology is therefore not known with absolute confidence. This data uncertainty is propagated to the load through the computation made. If it is neither present in the model nor not considered by the adaptation process, then the adaptation

1 process may make suboptimal reconfiguration decision.

2 The literature refers to systems which provide adaptation capabilities as dynamically  
3 adaptive systems (DAS). One challenge in the grid is the phase difference between the  
4 monitoring frequency and the time for actions to have measurable effects. Action with  
5 no immediate measurable effects are named **delayed action**. On the one hand, an  
6 incident should be detected in the next minutes. On the other hand, a reconfiguration  
7 action can take up to several hours. For example, when a tree falls on a cable and cuts  
8 it during a storm, the grid manager should be noticed in real time. The reconfiguration  
9 of the grid, to reconnect as many people as possible before replacing the cable, is done  
10 by technicians who need to use their cars to go on the reconfiguration places. In a fully  
11 autonomous adaptive system, the reasoning process should be considered the ongoing  
12 actions to avoid repeating decisions.

13 *Problematic* **Data uncertainty and delayed actions are not specific to smart**  
14 **grids.**

15 First, data are, almost by definition, uncertain and developers always work with  
16 estimates. Hardware sensors have by construction a precision that can vary accord-  
17 ing to the current environment in which they are deployed. A simple example is the  
18 temperature sensor that provides a temperature with precision to the nearest degree.  
19 Software sensors approximate also values from these physical sensors, which increases  
20 the uncertainty. For example, CPU usage is computed counting the cycle used by a  
21 program. As stated by Intel, this counter is not error-prone<sup>1</sup>.

22 Second, it always exists a delay between the moment where a suboptimal state is  
23 detected by the adaptation process and the moment where the effects of decisions taken  
24 are measured. This delayed is due to the time needed by a computer to process data  
25 and, eventually, to send orders or data through networks. For example, migrating a  
26 virtual machine from a server to another one can take several minutes.

27 **Through this thesis, I argue that this data uncertainty and this delay**  
28 **cannot be ignored for all dynamic adaptive systems.** To know if the data un-  
29 certainty should be considered, stakeholders should wonder **if this data uncertainty**

---

<sup>1</sup><https://software.intel.com/en-us/itc-user-and-reference-guide-cpu-cycle-counter>

1 **affects the result of their reasoning process, like adaptation.** Regarding delayed  
2 action, they should verify **if the frequency of the monitoring stage is lower than**  
3 **the time of action effects to be measurable.** These characteristics are common  
4 to smart grids, cloud infrastructure or cyber-physical systems in general.

5 *Challenge* These problematics come with different challenges concerning the represen-  
6 tation of the knowledge for DAS. The global challenge address by this thesis is: **how**  
7 **to represent the uncertain knowledge allowing to efficiently query it and to**  
8 **represent ongoing actions in order to improve adaptation processes?**

9 *Vision* **This thesis defends the need for a unified modeling framework which**  
10 **includes, despite all traditional elements, temporal and uncertainty as first-**  
11 **class concepts.** Therefore, a developer will be able to abstract information related to  
12 the adaptation process, the environment as well as the system itself.

13 Concerning the adaptation process, the framework should enable abstraction of the  
14 actions, their context, their impact, and the specification of this process (requirements  
15 and constraints). It should also enable the abstraction of the system environment and its  
16 behavior. Finally, the framework should represent the structure, behavior and specifi-  
17 cation of the system itself as well as the actuators and sensors. All these representations  
18 should integrate the data uncertainty existing.

19 *Contributions* Towards this vision, this document presents two approaches: a temporal  
20 context model and a language for uncertain data.

21 The temporal context model allows abstracting past, ongoing and future actions  
22 with their impacts and context. First, a developer can use this model to know what the  
23 ongoing actions, with their expect future impacts on the system, are. Second, she/he  
24 can navigate through past decisions to understand why they have been made when they  
25 have led to a sub-optimal state.

26 The language, named Ain'tea, integrates data uncertainty as a first-class concept. It  
27 allows developers to attach data with a probability distribution which represents their  
28 uncertainty. Plus, it mapped all arithmetic and boolean operators to uncertainty prop-  
29 agation operations. And so, developers will automatically propagate the uncertainty

1 of data without additional effort, compared to an algorithm which manipulates certain  
2 data.

3 *Validation* Each contribution has been evaluated separately. The language has been  
4 evaluated through two axes: its ability to detect errors at development time and its  
5 expressiveness. Ain'tea can detect errors in the combination of uncertain data earlier  
6 than state-of-the-art approaches. The language is also as expressive as current ap-  
7 proaches found in the literature. Moreover, we use this language to implement the load  
8 approximation of a smart grid furnished by an industrial partner, Creos S.A.<sup>2</sup>.

9 The context model has been evaluated through the performance axis. The disser-  
10 tation shows that it can be used to represent the Luxembourg smart grid. The model  
11 also provides an API which enables the execution of query for diagnosis purpose. In  
12 order to show the feasibility of the solution, it has also been applied to the use case  
13 provided by the industrial partner.

14 **Keywords:** dynamically adaptive systems, knowledge representation, model-driven  
15 engineering, uncertainty modeling, time modeling

---

<sup>2</sup>Creos S.A. is the power grid manager of Luxembourg. <https://www.creos-net.lu>

---

# 1 Table of Contents

---

2	<b>1 Introduction</b>	<b>1</b>
3	1.1 Use case: Luxembourg smart grid . . . . .	2
4	1.2 General background . . . . .	2
5	<b>2 TKM: a temporal knowledge model</b>	<b>3</b>
6	2.1 Introduction . . . . .	4
7	2.2 Knowledge formalization . . . . .	4
8	2.2.1 Formalization of the temporal axis . . . . .	5
9	2.2.2 Formalism . . . . .	6
10	2.2.3 Application on the use case . . . . .	10
11	2.3 Modeling the knowledge . . . . .	14
12	2.3.1 Parent element: <i>TimedElement</i> class . . . . .	15
13	2.3.2 Knowledge metamodel . . . . .	15
14	2.3.3 Context metamodel . . . . .	17
15	2.3.4 Requirement metamodel . . . . .	18
16	2.3.5 Action metamodel . . . . .	18
17	<b>Bibliography</b>	<b>iii</b>





1

---

## 2 Introduction

---

### 3 Contents

4	1.1 Use case: Luxembourg smart grid . . . . .	2
5		
6	1.2 General background . . . . .	2
7		
8		

---

10     **Abstract:** *Model-driven engineering methodology and dynamically adaptive systems*  
11 *approach are combined to tackle new challenges brought by systems nowadays. After*  
12 *introducing these two software engineering techniques, I give one example of such sys-*  
13 *tems: the Luxembourg smart grid. I will also use this example to highlight two of the*  
14 *problematics: uncertainty of data and delays in actions. Among the different challenges*  
15 *which are implied by them, I present the global one addressed by the vision defended in*  
16 *this thesis: modeling of temporal and uncertain data. This global challenge can be ad-*  
17 *dressed by splitting up in several ones. I present two of them, which are directly tackled*  
18 *by two contributions presented in this thesis.*

## **1.1 Use case: Luxembourg smart grid**

Should contain: - veg iqu grqeub

## **1.2 General background**

should contain: - MDE / metamodel / model - DAS

---

## TKM: a temporal knowledge model to represent actions, their contexts and their impacts

---

### Contents

---

<b>2.1 Introduction</b>	4
<b>2.2 Knowledge formalization</b>	4
<b>2.3 Modeling the knowledge</b>	14

---

**Abstract:** *The evolving complexity of adaptive systems impairs our ability to deliver anomaly-free solutions. Fixing these systems require a deep understanding on the reasons behind decisions which led to faulty or suboptimal system states. Developers thus need diagnosis support that trace system states to the previous circumstances –targeted requirements, input context– that had resulted in these decisions. However, the lack of efficient temporal representation limits the tracing ability of current approaches. To tackle this problem, we first propose a knowledge formalism to define the concept of a decision. Second, we describe a novel temporal data model to represent, store and query decisions as well as their relationship with the knowledge (context, requirements, and actions). We validate our approach through a use case based on the smart grid at Luxembourg. We also demonstrate its scalability both in terms of execution time and consumed memory.*

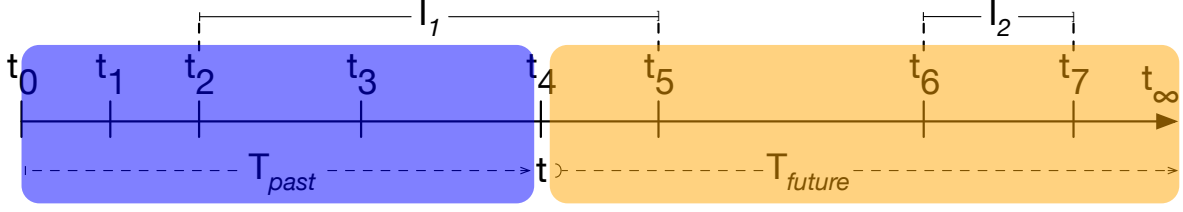


Figure 2.1: Time definition used for the knowledge formalism

## 2.1 Introduction

should define: decision, action, context, knowledge

## 2.2 Knowledge formalization

As discussed previously, I consider **knowledge** to be the association of **context** information, **requirements**, and **action** information, all in one global and unified model. While **context** information captures the state of the system environment and its surroundings, the system **requirements** define the constraints that the system should satisfy along the way. The **actions**, on the other hand, are means to reach the goals of the system.

In this section, I provide a formalization of the **knowledge** used by adaptation processes based on a temporal graph. Indeed, due to the complexity and interconnectivity of system entities, graph data representation seems to be an appropriate way to represent the **knowledge**. Augmented with a temporal dimension, temporal graphs are then able to symbolize the evolution of system entities and states over time. We benefit from the well-defined graph manipulation operations, namely temporal graph pattern matching and temporal graph relations to represent the traceability links between the **decisions** made and their **circumstances**.

Before describing this formalism, I describe the semantic used for the temporal axis. Then, I exemplify the knowledge formalism using the Luxembourg smart grid use case.

### 2.2.1 Formalization of the temporal axis

The formalism describe below has been made with two goals in mind. First, the definition of the time space should allow the distinction between past and future. Doing this distinction enable the differentiation between measured data and estimated (or predicted data). Second, it should permit the definition of the life cycle of an element of the **knowledge**, which can be seen as a succession of states with a validity period that should not overlap each other.

Time space  $T$  is considered as an ordered discrete set of time points non-uniformly distributed. As depicted in Figure 2.1, this set can be divided into 3 different subsets  $T = T_{past} \cup \{t\} \cup T_{future}$ , where:

- $T_{past}$  is the sub-domain  $\{t_0; t_1; \dots; t_{current-1}\}$  representing graph data history starting from  $t_0$ , the oldest point, until current time,  $t$ , excluded.
- $\{t\}$  is a singleton representing the current time point
- $T_{future}$  is sub-domain  $\{t_{current+1}; \dots; t_{\infty}\}$  representing future time points

The three domains depend completely on the current time  $\{t\}$  as these subsets slide as time passes. At any point in time, these domains never overlap:  $T_{past} \cap \{t\} = \emptyset$ ,  $T_{future} \cap \{t\} = \emptyset$ , and  $T_{past} \cap T_{future} = \emptyset$ . The definition of these three subsets reaches the first goal.

In addition, there is a right-opened time interval  $I \in T \times T$  as  $[t_s, t_e)$  where  $t_e - t_s > 0$ . In English words, it means that the interval cannot represent a single time point and should follow the time order. For any  $i \in I$ ,  $start(i)$  denotes its lower bound and  $end(i)$  its upper bound. As detailed in Section 2.2.2, these intervals are used to define the validity period for each node of the graph.

Figure 2.1 displays an example of a time space  $T_1 = \{t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$ . Here, the current time is  $t = t_4$ . According to the definition of the past subset ( $T_{past}$ ) and the future one ( $T_{future}$ ), there is:  $T_{past1} = \{t_0, t_1, t_2, t_3\}$  and  $T_{future1} = \{t_5, t_6, t_7\}$ . Two intervals have been defined on  $T_1$ , namely  $I_1$  and  $I_2$ . The first one starts at  $t_2$  and ends at  $t_5$  and the last one is defined from  $t_6$  to  $t_7$ . As shown with  $I_1$ , an interval could be defined on different subsets, here it is on all of them ( $T_{past}$ ,  $t$ , and  $T_{future}$ ).

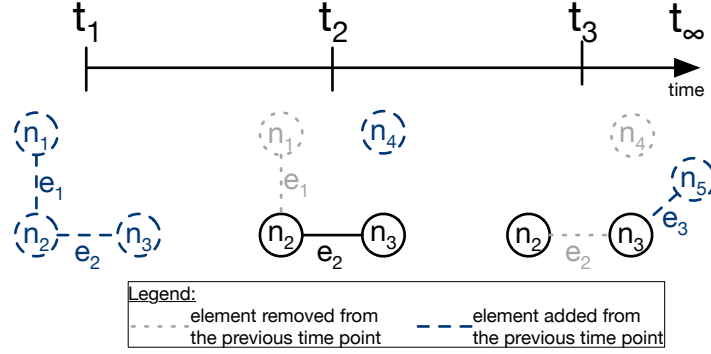


Figure 2.2: Evolution of a temporal graph over time

## 2.2.2 Formalism

**Graph definition** First, let  $K$  be an adaptive process over a system **knowledge** represented by a graph such as  $K = (N, E)$ , comprising a set of nodes  $N$  and a set of edges  $E$ . Nodes represent any element of the knowledge (context, actions, *etc.*) and edges represent their relationships. Nodes have a set of attribute values. An attribute value has a type (numerical, boolean,  $\dots$ ). Every relationship  $e \in E$  can be considered as a couple of nodes  $(n_s, n_t) \in N \times N$ , where  $n_s$  is the source node and  $n_t$  is the target node.

**Adding the temporal dimension** In order to augment the graph with a temporal dimension, the relation  $V^T$  is added. So now the knowledge  $K$  is defined as a temporal graph such as  $K = (N, E, V^T)$ .

A node is considered valid either until it is removed or until one of its attributes value changes. In the latter case, a new node with the updated value is created. Whilst, an edge is considered valid until either its source node and target node is valid, or until the edge itself is removed. Otherwise, nodes and edges are considered invalid. The temporal validity relation is defined as  $V^T : N \cup E \rightarrow I$ . It takes as a parameter a node or an edge ( $k \in N \cup E$ ) and returns a time interval ( $i \in I$ , *cf.* Section 2.2.1) during which the graph element is valid.

Figure 2.2 shows an example of a temporal graph  $K_1$  with five nodes ( $n_1, n_2, n_3, n_4$ , and  $n_5$ ) and three edges ( $e_1, e_2$ , and  $e_3$ ) over a lifecycle from  $t_1$  to  $t_3$ . In this

way,  $K_1$  equals to  $(\{n_1, n_2, n_3, n_4, n_5\}, \{e_1, e_2, e_3\}, V_1^T)$ . Let's assume that the graph is created at  $t_1$ . As  $n_1$  is modified at  $t_2$ , its validity period starts at  $t_1$  and ends at  $t_2$ :  $V_1^T(n_1) = [t_1, t_2)$ .  $n_2$  and  $n_3$  are not modified; their validity period thus starts at  $t_1$  and ends at  $t_\infty$ :  $V_1^T(n_2) = V_1^T(n_3) = [t_1, t_\infty)$ . Regarding the edges, the first one,  $e_1$ , is between  $n_1$  and  $n_2$  and the second one,  $e_2$  from  $n_2$  to  $n_3$ . Both are created at  $t_1$ . As  $n_1$  is being modified at  $t_2$ , its validity period goes from  $t_1$  to  $t_2$ :  $V_1^T(e_1) = [t_1, t_2)$ .  $e_2$  is deleted at  $t_3$ . Its validity period is thus equal to:  $V_1^T(e_2) = [t_1, t_3)$ .

**Lifecycle of a knowledge element** One node represents the state of exactly one knowledge element during a period named the validity period. The lifecycle of a knowledge element is thus modeled by a unique set of nodes. By definition, the validity periods of the different nodes cannot overlap. A same time period cannot be represented by two different nodes, which could create inconsistency in the temporal graph.

To keep track of this knowledge element history, the  $Z^T$  relation is added to the graph formalism:  $K = (N, E, V^T, Z^T)$ . It serves to trace the updates of a given knowledge element at any point in time. This relation can also be seen as a temporal identity function which takes as parameters a given node  $n \in N$  and a specific time point  $t \in T$ , and returns the corresponding node at that point. Formally,  $Z^T : N \times T \rightarrow N$ .

In order to consider this new relation in the example presented in Figure 2.2, the definition of  $K_1$  is modified to  $K_1 = (\{n_1, n_2, n_3, n_4, n_5\}, \{e_1, e_2, e_3\}, V_1^T, Z_1^T)$ . In Figure 2.2, let's imagine that  $n_1$ ,  $n_4$ , and  $n_5$  represent the same knowledge element  $k_e$ . The lifecycle of  $k_e$  is thus:

- $n_1$  for period  $[t_1, t_2)$ ,
- $n_4$  for period  $[t_2, t_3)$ ,
- $n_5$  for period  $[t_3, t_\infty)$ .

Let  $t'_1$  be a timepoint between  $t_1$  and  $t_2$ . When one wants to resolve the node representing the knowledge element at  $t'_1$ , she or he gets  $n_1$  node, no matter of the node input ( $n_1$ ,  $n_4$ , or  $n_5$ ):  $Z_1^T(n_4, t_1) = n_1$ . On the other hand, applying the same relation with another node ( $n_2$  or  $n_3$ ) returns another node. For example, if  $n_2$  and  $n_3$  do not belongs to the same knowledge element, then it will return the node given as input, for example  $Z_1^T(n_2, t_1) = n_2$ .

1 **Knowledge elements stored in nodes** Nodes are used to store the different knowl-  
2 edge elements: context, requirements and actions. The set of nodes  $N$  is thus split in  
3 three subset:  $N = C \cup R \cup A$  where  $C$  is the set of nodes which store context informa-  
4 tion,  $R$  a set of nodes for requirement information and  $A$  the set of nodes for actions  
5 information.

6 Actions define a process that indirectly impact the context: they will change the  
7 behavior of the system, which will be reflected on the context information. Require-  
8 ments are also processes that are continuously run over the system in order to check the  
9 specifications. Here, the purpose of the  $A$  and  $R$  subset is not to store these processes  
10 but to list them. It can be thought as a catalogue of actions and requirements, with  
11 their history.

12 Using a high level overview, these processes can be depicted as: taking the knowl-  
13 edge as input, perform task, and modify this knowledge as output. As detailed in the  
14 next two paragraphs, they can be formalized by relations.

15 **Temporal queries for requirements** At the current state, the formalism of the  
16 knowledge  $K$  do not contain any information regarding the requirement processes. To  
17 overcome this, system requirements processes  $R_P$  are added such as  $K = (N, E, V^T, Z^T,$   
18  $R_P)$ .  $R_P$  is a set of patterns  $P_{[t_j, t_k]}(K)$  and queries  $Q$  over these patterns:  $R_P = P \cup Q$ .

19  $P_{[t_j, t_k]}$  denotes a temporal graph pattern, where  $t_j$  and  $t_k$  are the lower and upper  
20 bound of the time interval respectively. The time interval can be either fixed (absolute)  
21 or sliding (relative). Each element of the pattern should be valid for at least one time  
22 point:  $\forall p \in P_{[t_j, t_k]}, V^T(e) \cap [t_j, t_k] \neq \emptyset$ . Patterns can be seen as temporal subgraph  
23 of  $K$ , with a time limiting constraint coming in the form of a time interval. Temporal  
24 graph queries  $Q$  consist commonly of two parts: (i) path description to traverse the  
25 graph nodes, at both structural and temporal dimensions; (ii) arithmetic expressions  
26 on nodes, edges, and attribute values.

27 **Temporal relations for actions** Like for  $R_P$ , the knowledge  $K$  needs to be aug-  
28 mented with the action processes  $A_P$ :  $K = (N, E, V^T, Z^T, R_P, A_P)$ . Actions processes  
29  $A_P$  can be regarded as a set of relations or isomorphisms mapping a source temporal  
30 graph pattern  $P_{[t_j, t_k]}$  to a target one  $P_{[t_l, t_m]}$ ,  $A_P : K \times I \rightarrow K \times I$ .



1 The left-hand side of the relation depicts the temporal graph elements over which  
 2 an action is applied. Every relation may have a set of application conditions. They  
 3 describe the circumstances under which an action should take place. These application  
 4 conditions are either positive, should hold, or negative, should not hold. Application  
 5 conditions come in the form of temporal graph invariants. The side effects of these  
 6 actions are represented by the right-hand side.

7 Finally, we associate to  $A_P$  a temporal function  $E_{A_P}$  to determine the time interval  
 8 at which an action has been executed. Formally,  $X : A \rightarrow I$ .

9 **Temporal relations for decisions** Finally, the knowledge formalism needs to in-  
 10 clude the last, but not the least, element: decisions made by the adaptation,  $K = (N,$   
 11  $E, V^T, Z^T, R_P, A_P, D)$  While the source of relations in  $D$  represents the state before  
 12 the execution of an action, the target shows its impact on the **context**. Its intent is  
 13 **to trace back impacts of actions execution to the decisions they originated**  
 14 **from.**

15 A decision present in  $D$  is defined as a set of executed actions, *i.e.*, a subset of  $A_P$ .  
 16 Formally,  $D = \{ A_D \cup R_D \mid A_D \subseteq A_P, R_D \subseteq R_P \}$ . We assume that each action should  
 17 result from one decision:  $\forall a \in A, \forall d1, d2 \in D \mid a \in d1 \wedge a \in d2 \rightarrow d1 = d2$ .

18 The temporal function  $E_{A_P}$  is extended to decision in order to represent the execu-  
 19 tion time:  $E_{A_P} : (A \cup D) \rightarrow I$ . For decision, the lower bound of the interval correspond  
 20 to the lowest bound of the action execution intervals. Following the same principle, the  
 21 upper bound of the interval correspond to the uppermost bound of the action execu-  
 22 tion intervals. Formally,  $\forall d \in D \rightarrow E_{A_P}(d) = [l, u)$ , where  $l = \min_{a \in A_d} \{E_{A_P}(a)[start]\}$  and  
 23  $u = \max_{a \in A_d} \{E_{A_P}(a)[end]\}$ .

24 **Sum up** Knowledge of an adaptive system can be formalism with a temporal graph  
 25 such as  $K = (N, E, V^T, Z^T, R_P, A_P, D)$ , wherein:

- 26 •  $N$  is a set of nodes to represent the different information (context, actions and  
 27 requirements)
- 28 •  $E$  is a set of edges with connect the different nodes,
- 29 •  $V^T$  is a temporal relation which defines the temporal validity of each elements,
- 30 •  $Z^T$  is a relation to track the history of each knowledge elements,

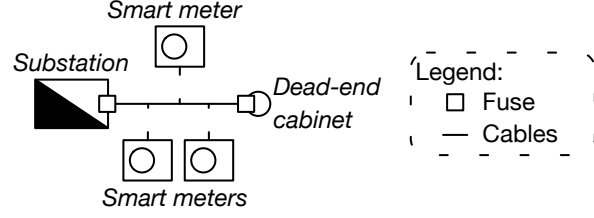


Figure 2.3: Simplified version of a smart grid

Figure 2.4: Representation of the smart grid context depicted in Figure 2.3

- 1 •  $R_P$  is a relation that define the different requirements processes,
- 2 •  $A_P$  is a relation that define the different action processes,
- 3 •  $D$  is a set of action processes that result from a same decision.

4 In the next section, we exemplify this formalism over our case study.

### 5 2.2.3 Application on the use case

6 The example presented in Section 1.1 contain too much detail to provide a readable  
7 and understandable example of the formalism. Below, an excerpt of it is thus presented  
8 in order to overcome this problem.

9 **Excerpt of a smart grid** Figure 2.3 shows a simplified version of a smart grid with  
10 one substation, one cable, three smart meters and one dead-end cabinet. Both the  
11 substation and the cabinet have a fuse. The meters regularly send consumption data  
12 at the same time. One requirement is considered for this example: minimizing the  
13 number of overloads. To achieve so, among the different actions, two actions are taken  
14 into account in this example: decreasing or increasing the amps limits of smart meters.

15 Let  $K_{SG}$  be the temporal graph that represents the knowledge of this adaptive  
16 system:  $K_{SG} = (N_{SG}, E_{SG}, V_{SG}^T, Z_{SG}^T, R_{P_{SG}}, A_{P_{SG}}, D_{SG})$ . Figure 2.4 shows the nodes  
17 and edges of this knowledge.

18 **Scenario** The system starts at  $t_0$  with the actions, the requirements and the context,  
19 which also include initial value for the consumption values. Meters send their values at  
20  $t_2$  and  $t_3$ . Based on these data, the load on cables and substation is computed. On  $t_2$ ,

1 an overload is detected on the cable, which break the requirement. At the same time  
 2 point, the system decides to reduce the load of all smart meters. The impact of these  
 3 actions will be measured at  $t_4$ , *i.e.*, the cable will not be overloaded from  $t_4$ .

4 **Description of  $N_{SG}$**   $N_{SG}$  is divided into three subset:  $C_{SG}$ ,  $R_{SG}$  and  $A_{SG}$ .  $R_{SG}$   
 5 contains one node,  $R_1$  in Figure 2.4, which represents the requirement of this example:  
 6  $R_{SG} = \{R_1\}$  Two nodes,  $A_1$  and  $A_2$ , belong to  $A_{SG}$ :  $A_{SG} = \{A_1, A_2\}$ . They represent  
 7 represent the two actions of this example, respectively decreasing and increasing amps  
 8 limits. Regarding the context  $C_{SG}$ , there is three nodes to represent the three smart  
 9 meters ( $M_1$ ,  $M_2$ , and  $M_3$ ), one for the substation ( $S_1$ ), two for the fuses ( $F_1$  and  $F_2$ ),  
 10 one for the dead-end cabinet ( $D_{C_1}$ ) and one node per consumption value received ( $V_i$ ):  
 11  $C_{SG} = \{M_1, M_2, M_3, S_1, F_1, F_2, D_{C_1}\} \cup \{V_i | i \in [1..9]\}$ .

12 According to the scenario, all nodes are created at  $t_0$  and are never modified, except  
 13 for nodes to store consumption values. Therefore, their validity period starts at  $t_0$   
 14 and never ends:  $\forall n \in A_{SG} \cup R_{SG} \cup \{M_1, M_2, M_3, S_1, F_1, F_2, D_{C_1}\}, V_{SG}^T(n) = [t_0, t_\infty)$ .  
 15 Considering the consumption values, all the nodes represent the history of the values  
 16 for the three smart meters. In other words, there is three knowledge element: the  
 17 consumption measured for each meter. Let  $C_i$  notes the consumption measured by the  
 18 smart meter  $M_i$ . As shown in Figure 2.4, there is:

- 19 •  $C_1$  of  $M_1$  is represented by  $\{V_1, V_4, V_7\}$ ,
- 20 •  $C_2$  of  $M_2$  is represented by  $\{V_2, V_5, V_8\}$ ,
- 21 •  $C_3$  of  $M_3$  is represented by  $\{V_3, V_5, V_9\}$ .

22 Taking  $C_2$  as example,  $V_2$  is the initial consumption value, replaced by  $V_5$  at  $t_1$ , itself  
 23 replaced by  $V_8$  at  $t_2$ . Applying the  $V_{SG}^T$  on these different values, results are thus:

- 24 •  $V_{SG}^T(V_2) = [t_0, t_1)$ ,
- 25 •  $V_{SG}^T(V_5) = [t_1, t_2)$ ,
- 26 •  $V_{SG}^T(V_8) = [t_2, t_\infty)$ .

27 These validity periods are shown in Figure 2.5a. As meters send the new consumption  
 28 values at the same time, this example can be also applied to  $C_1$  and  $C_3$ .

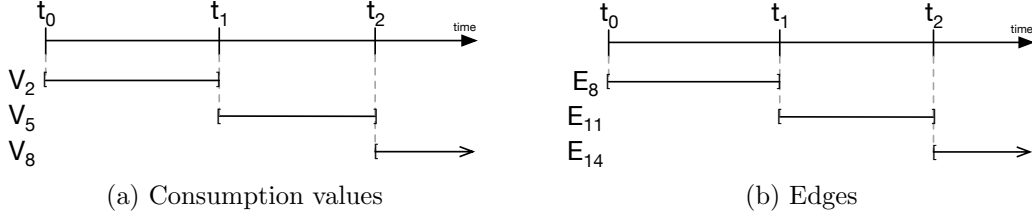


Figure 2.5: Validity periods of the consumptions values and their edges to the smart meter  $M_2$

From these validity period, the  $Z_{SG}^T$  can be used to navigate to the different values over time. Let's continue with the same example,  $C_2$ . In order to get the evolution of the consumption value  $C_2$ , given the initial one, one will use the  $Z_{SG}^T$  relation:

- $Z_{SG}^T(V_2, t_{s1}) = V_2$ , where  $t_0 \leq t_{s1} < t_1$
- $Z_{SG}^T(V_2, t_{s2}) = V_5$ , where  $t_1 \leq t_{s2} < t_2$
- $Z_{SG}^T(V_2, t_{s3}) = V_8$ , where  $t_2 \leq t_{s3} < t_\infty$ .

**Description of  $E_{SG}$**  In this example, edges are used to store the relationships between the different context elements. For example, the edge between the substation  $S_1$  and the fuse  $F_1$  allow to represent the fact that the fuse is physically inside the substation. Another example, edges between the cable  $C_1$  and the meters  $M_1$ ,  $M_2$  and  $M_3$  represent the fact that these meters are connected to the smart grid through this cable.

One may consider that relations (validity,  $Z^T$ , decisions, action processes and requirements processes) will be stored as edges. But, this decision is let to the implementation part of this formalism.

In our model, only consumption values ( $V_i$  nodes) are modified. Plus, since the scenario do not imply other edges modifications, only those between meters and values are modified. The edge set contains thus sixteen edges:  $E_{SG} = \{E_i \mid i \in [1..16]\}$ .

By definition, the unmodified edges have a validity period starting from  $t_0$  and never ends:  $\forall i \in [1..7], V_{SG}^T(E_i) = [t_0, t_\infty)$ . The history of the three knowledge elements that represent consumption values do not only impact the nodes which represent the values but also the edges between those nodes and the meters ones:

- $C_1$  impacts edges between  $M_1$  and  $V_1$ ,  $V_4$ , and  $V_7$ , i.e.,  $\{E_8, E_{11}, E_{14}\}$ ,

- 1 •  $C_2$  impacts edges between  $M_2$  and  $V_2$ ,  $V_5$ , and  $V_8$ , *i.e.*,  $\{E_9, E_{12}, E_{15}\}$ ,
- 2 •  $C_3$  impacts edges between  $M_3$  and  $V_3$ ,  $V_6$ , and  $V_9$ , *i.e.*,  $\{E_{10}, E_{13}, E_{16}\}$ .

3 Continuing with  $C_2$  as example, the initial edge value is  $E_8$  from  $t_0$ , which is replaced  
 4 by  $E_{11}$  from  $t_1$ , itself replaced by  $E_{14}$  from  $t_2$ . The validity relation, applied on these  
 5 edges, thus returns:

- 6 •  $V_{SG}^T(E_8) = [t_0, t_1) = V_{SG}^T(V_2)$ ,
- 7 •  $V_{SG}^T(E_{11}) = [t_1, t_2) = V_{SG}^T(V_5)$ ,
- 8 •  $V_{SG}^T(E_{14}) = [t_2, t_\infty) = V_{SG}^T(V_8)$ ,

9 These validity periods are depicted in Figure 2.5b. As they are driven by those of  
 10 consumption values ( $V_2$ ,  $V_5$ , and  $V_8$ ), they are equals.

11 As for nodes, the  $Z_{SG}^T$  relation can navigate over time through these values. For  
 12 example, to get the history of the edges between the consumption value  $C_2$  and the  
 13 meter represented by  $M_2$ , one can apply the  $Z_{SG}^T$  relation as following:

- 14 •  $Z_{SG}^T(E_8, t_{s1}) = E_8$ , where  $t_0 \leq t_{s1} < t_1$ ,
- 15 •  $Z_{SG}^T(E_8, t_{s2}) = E_8$ , where  $t_1 \leq t_{s1} < t_2$ ,
- 16 •  $Z_{SG}^T(E_8, t_{s3}) = E_8$ , where  $t_2 \leq t_{s1} < t_\infty$ .

17 **Description of  $R_{PSG}$**  The requirement calls for minimizing overloads. It means that  
 18 when the system detects at least one overload, for example in cables, it will take counter  
 19 actions. As the system has prediction capabilities, it will not only check is there is one  
 20 at the current time  $t$  but also if one will come in the next hour. The pattern will be  
 21 defined as follow:  $P_{[t, t+15min]}$ . To determine if there is an overload, the system needs to  
 22 know: the current and future consumption, the current and future topology. The last  
 23 one is used to compute the loads from the consumption (cf. Section 1.1).

24 Let's consider that time points are regular and there is one every 15 minutes and  
 25 that current time is  $t_0$ . The pattern,  $P_{[t_0, t_1]}$ , will thus contain all nodes that are valid  
 26 between  $t_0$  and  $t_1$  (included):

- 27 • all topology nodes between:  $\{S_1, C_1, F_1, F_2, D_{C_1}, M_1, M_2, M_3\}$
- 28 • all consumption values between:  $\{V_i \mid i \in [1..6]\}$ ,

- all edges that connected these nodes:  $\{E_i \mid i \in [1..13]\}$

From these values, the loads is computed and the system checks that none will exceed the capacity of the infrastructure (cables, substations, cabinets).

**Description of  $A_{P_{SG}}$**  Now, let us assume that the execution of  $R_{P_{SG}}$  detects an overload on the cable ( $C_1$ ) at  $t_0$ . The system decides to reduce the amps limits, and thus the load, on the three meters. The action  $A_1$  (decreasing amps limits) is thus executed three times: one time per meter. For each of these action, the input context will correspond to the pattern used by the requirement relation:  $P_{[t_0, t_1]}$ . The output context will contain the predicted values after the actions have been executed. Here, the actions are executed in parallel and their execution time is in seconds. So the impact will be visible from  $t_1$ . So the output pattern contain the three values at  $t_1$ :  $P_{[t_1, t_1]}$ . In summary:

- Action 1:  $A_{P_1} : P_{[t_0, t_1]} \rightarrow P_{[t_1, t_1]}$ ,
- Action 1:  $A_{P_2} : P_{[t_0, t_1]} \rightarrow P_{[t_1, t_1]}$ ,
- Action 1:  $A_{P_3} : P_{[t_0, t_1]} \rightarrow P_{[t_1, t_1]}$ .

**Description of  $D_{SG}$**  Following the scenario, there is one decision,  $D_{SG_1}$ , which try to achieve the requirement  $R_1$  by executing the actions  $A_1$ :  $A_{P_1}$ ,  $A_{P_2}$ , and  $A_{P_3}$ . Then, here the decision is equals to:  $D_{SG_1} = \{R_1, A_{P_1}, A_{P_2}, A_{P_3}\}$ .

**Summrarize** Through this section, I explified how the formalism can be used to define an adaptation decision on a smart grid system. As the decision contains information about the circumstances and the impact, one may use it to debug the process and/or try to explain the behavior of such systems.

## 2.3 Modeling the knowledge

In order to simplify the diagnosis of adaptive systems, this thesis proposes a novel **metamodel** that combines, what I call, design elements and runtime elements. Design elements abstract the different elements involved in **knowledge** information to assist the specification of the adaptation process. Runtime elements instead, represent the data collected by the adaptation process during its execution. In order to maintain

1 the consistency between previous design elements and newly created ones, instances  
2 of design elements (*e.g.*, actions) can be either added or removed. Modifying these  
3 elements would consist in removing existing elements and creating new ones. Combining  
4 design elements and runtime elements in the same model helps not only to acquire  
5 the evolution of system but also the evolution of its structure and specification (e.g.  
6 evolution of the requirements of the system). Design time elements are depicted in gray  
7 in the Figures 2.6– 2.9. Note that, this thesis does not address how runtime information  
8 is collected.

9 For the sake of modularity, the **metamodel** has been split into four packages: Knowl-  
10 edge, Context, Requirement and Action. All the classes of these packages have a com-  
11 mon parent class that adds the temporality dimension: *TimedElement* class. Before  
12 describing the Knowledge (core) package, I detail this element. Then, I introduce in  
13 more details the other three packages used by the Knowledge package: Context, Re-  
14 quirement, and Action. In below sections, I use "*Package::Class*" notation to refer to  
15 the provenance of a class. If the package is omitted, then the provenance package is  
16 this one described by the figure or text.

### 17 2.3.1 Parent element: *TimedElement* class

18 I assume that all the classes in the different packages extend a *TimedElement* class.  
19 This class contains three methods: *startTime*, *endTime*, and *modificationsTime*. The  
20 first two methods allow accessing the validity interval bounds defined by the previously  
21 discussed  $V^T$  relation. The last method resolves all the timestamps at which an element  
22 has been modified: its history. This method is the implementation of the relation  $Z^T$   
23 described in our formalism (cf. Section 2.2.2).

### 24 2.3.2 Knowledge metamodel

25 In order to enable interactive diagnosis of adaptive systems, traceability links be-  
26 tween the decisions made and their circumstances should be organized in a well-structured  
27 representation. In what follows, I introduce how the knowledge **metamodel** helps to de-  
28 scribe decisions, which are linked to their goals and their context (input and impact).  
29 Figure 2.6 depicts this **metamodel**.

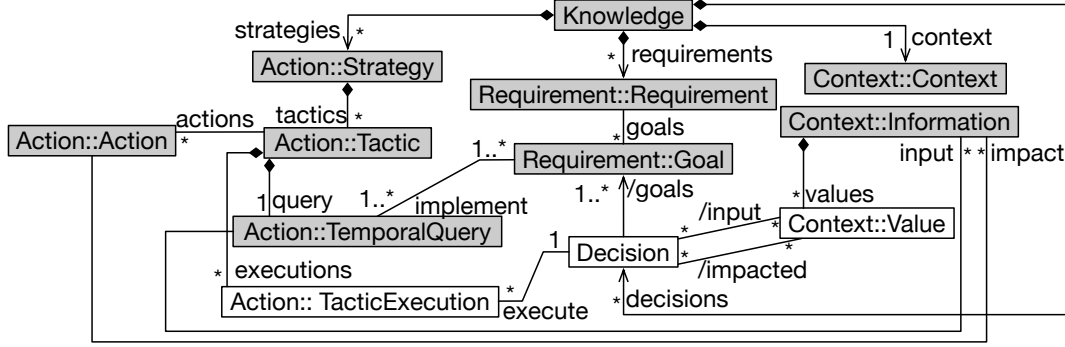


Figure 2.6: Excerpt of the knowledge metamodel

1 Knowledge package is composed of a *context*, a set of *requirements*, a set of *strategies*,  
 2 and a set of *decisions*. A decision can be seen as the output of the Analyze and Plan  
 3 steps in the **Monitor, Analyze, Plan, and Execute over knowledge (MAPE-k)** loop.

4 **MAPE-k** is fun!

5 Decisions comprise target *goals* and trigger the execution of one *tactic* or more. A  
 6 decision has an *input* context and an *impacted* context. The context impacted by a deci-  
 7 sion (*Decision.impact*) is a derived relationship computed by aggregating the impacts  
 8 of all actions belonging to a decision (see Fig. 2.9). Likewise, the *input* relationship is  
 9 derived and can be computed similarly. In the smart grid example, a decision can be  
 10 formulated (in plain English) as follows: since the district D is almost overloaded (*input*  
 11 *context*), we reduce the amps limit of greedy consumers using the “*reduce amps limit*”  
 12 *action* in order to reduce the load on the cable of the district (*impact*) and satisfy the  
 13 “*no overload*” policy (*requirement*).

14 As all the elements inherit from the *TimedElement*, we can capture the time at  
 15 which a given decision and its subsequent actions were executed, and when their im-  
 16 pact materialized, *i.e.*, measured. Thanks to this metamodel representation, we can  
 17 apprehend the possible causes behind malicious behavior by navigating from the con-  
 18 text values to the decisions that have impacted its value (*Property.expected.impact*) and  
 19 the goals it was trying to reach (*Decision.goals*). In Section ??, we present an example  
 20 of interactive diagnosis queries applied to the smart grid use case.



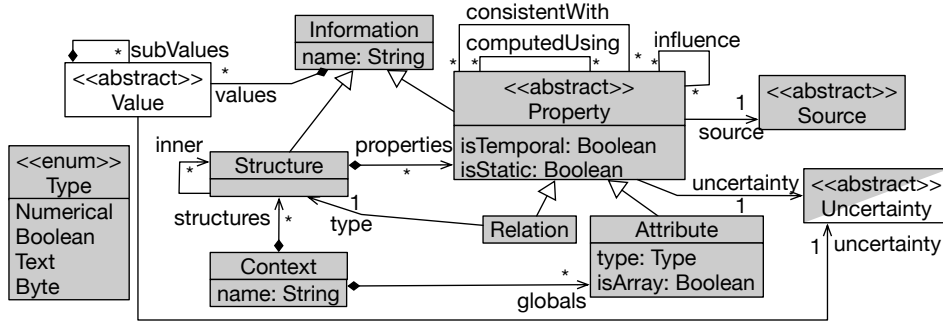


Figure 2.7: Excerpt of the context metamodel

### 2.3.3 Context metamodel

Context models structure context information acquired at runtime. For example, in a smart-grid system, the context model would contain information about smart-grid users (address, names, etc.) resource consumption, etc..

An excerpt of the context model is depicted in Figure 2.7. We propose to represent the context as a set of structures (*Context.structures*) and global attributes (*Context.globals*). A structure can be viewed as a C-structure with a set of properties (*Property*): attributes (*Attribute*) or relationships (*Relation*). A structure may contain other nested structures (*Structure.inner*). Structures and properties have values. They correspond to the nodes described in the formalization section (*cf.* Section ??). The connection feature described in Section ?? is represented thanks to three recursive relationships on the *Property* class: *consistentWith*, *computedUsing* and *influence*. Additionally, each property has a source (*Source*) and an uncertainty (*Uncertainty*). It is up to the stakeholder to extend data with the appropriate source: measured, computed, provided by a user, or by another system (*e.g.*, weather information coming from a public API). Similarly, the uncertainty class can be extended to represent the different kinds of uncertainties. Finally, a property can be either historic or static.

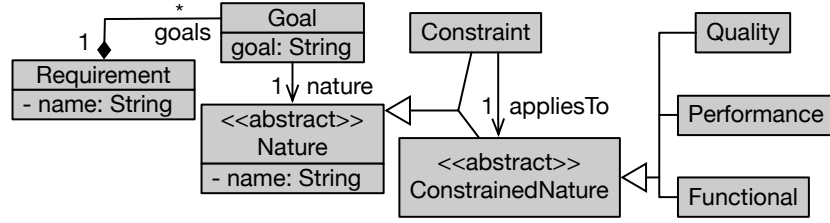


Figure 2.8: Requirement metamodel

### 2.3.4 Requirement metamodel

As different solutions to model system requirements exist (*e.g.*, KAOS [?], i\* [?] or Tropos [?]), in this metamodel, we abstract their shared concepts. Our requirement model, depicted in Figure 2.8, represents the *requirement* as a set of *goals*. Each goal has a *nature* and a textual specification. The nature of the goals adheres to the four categories of requirements presented in Section ???. We may use one of the existing requirements modeling languages (*e.g.*, RELAX) to define the semantics of the requirements. Since the requirement model is composed solely of design elements, we may rely on static analysis techniques to infer the requirement model from existing specifications. The work of Egyed [?] is one solution among others. This work is out of the scope of the paper and envisaged for future work.

In our guidance example, the requirement model may contain a **balanced resource distribution** requirement. It can be split into different goals: (i) *no overload*, (ii) *no production lack*, (iii) *no production loss*.

### 2.3.5 Action metamodel

Similar to the requirements metamodel, the actions metamodel also abstracts main concepts shared among existing solutions to describe adaptation processes and how they are linked to the context. Figure 2.9 depicts an excerpt of the action metamodel. We define a strategy as a set of tactics (*Strategy*). A tactic contains a set of actions (*Action*). A tactic is executed under a precondition represented as a temporal query (*TemporalQuery*) and uses different data from the context as input. In future work, we will investigate the use of preconditions to schedule the executions order of the actions,

1 similarly to existing formalisms such as Stitch [?]. The query can be as complex as  
 2 needed and can navigate through the whole knowledge model. Actions have impacts  
 3 on certain properties, represented by the *impacted* reference.

4 The different executions are represented thanks to the *Execution* class. Each ex-  
 5 ecution has a status to track its progress and links to the impacted context val-  
 6 ues(*Execution.impactedValues*). Similarly, input values are represented thanks to the  
 7 *Execution.inputValues* relationship. An execution has *start* and *end* time. Not to con-  
 8 fuse with the *startTime* and *endTime* of the validity relation  $\mathcal{V}^T$ . Whilst the former  
 9 corresponds to the time range in which a value is valid, the *start* and *stop* time in the  
 10 class execution correspond to the time range in which an action or a tactic was being  
 11 executed. The start and stop attributes correspond to the relation  $\mathcal{X}$  (see Section ??).  
 12 These values can be derived based on the validity relation. They correspond to the  
 13 time range in which the status of the execution is "RUNNING". Formally, for every  
 14 execution node  $e$ ,  $\mathcal{X}(e) = (\mathcal{V}(e) \mid e.status = "RUNNING")$ .

15 Similarly to requirement models, it is possible to automatically infer design elements  
 16 of action models by statically analyzing actions specification. Since acquiring informa-  
 17 tion about tactics and actions executions happens at runtime, one way to achieve this is  
 18 by intercepting calls to actions executions and updating the appropriate action model  
 19 elements accordingly. This is out of the scope of this paper and planned for future  
 20 work.

