Mind Maps Upstream SysML v2 Diagrams

Pierre De Saqui-Sannes, Rob A. Vingerhoeds, Nasrine Damouche, Eric Razafimahazo, Ombeline Aïello, Maisa Cietto

Mind Maps Upstream SysML v2 Diagrams

Pierre de Saqui-Sannes
ISAE-SUPAERO, Université de Toulouse
Toulouse, France
pdss@isae-supaero.fr

Rob A. Vingerhoeds
ISAE-SUPAERO, Université de Toulouse
Toulouse, France
rob.vingerhoeds@isae-supaero.fr

Nasrine Damouche
ISAE-SUPAERO, Université de Toulouse
Toulouse, France
nasrine.damouche@isae-supaero.fr

Eric Razafimahazo
ISAE-SUPAERO, Université de Toulouse
Toulouse, France
eric.razafimahazo@isae-supaero.fr

Ombeline Aïello
ISAE-SUPAERO, Université de Toulouse
Toulouse, France
ombeline.aiello@isae-supaero.fr

Maisa Cietto
Universidade de São Paulo
São Carlos, SP, Brazil
maisa.cietto@usp.br

***Abstract*—Over the past two decades, the promoters of Model Based Systems Engineering (MBSE) have encouraged systems engineers to transition from document-centric approaches to model-based ones. Literacy of systems engineers in reading, sharing and elaborating models has therefore become an issue. Whatever the modeling language, elaboration of models is a highly complex intellectual process and SysML is no exception. Feedback from industry practitioners and MBSE lecturers suggests that developers of SysML models often stumble on the same problem: thinking about the system before modeling it in SysML. The authors of this paper propose to ease that elaboration process by using mind maps. With their graphic form and rather flexible way of organizing ideas, mind maps turn out to be a good candidate to help thinking about the system. Unlike approaches that directly switch from mind maps to SysML diagrams dedicated to one specific system, this paper introduces an intermediate step: mind maps first enable elaboration of diagrams patterns. The latter may in turn be instantiated on one or several systems. Without loss of generality, the proposed approach is step-wise illustrated on real-time systems monitored by software controllers. Patterns are proposed to cover need expression, requirement capture, use case driven analysis and design.**

***Index Terms*—Mind Maps, SysML v2, Real-Time Systems.**

I. Introduction

Over the past two decades, the promoters of Model Based Systems Engineering (MBSE) have encouraged systems engineers to transition from document-centric approaches to model-based ones. Literacy of systems engineers in reading, sharing and elaborating models has therefore become an issue. In this context, the standardization and tool support of graphic modeling languages - particularly, the Unified Modeling Language (UML [1]) and the Systems Modeling Language (SysML [2]) - have been presented as enablers of MBSE adoption in industry [3].

Nevertheless, recent position papers, internships offers and posts on social networks have indicated that many industry practitioners are still in the process of evaluating the pros and cons of MBSE. Several reasons may been found. From their experience as MBSE lecturers, the authors of this paper wish to highlight one of them: the difficulty to address a system by immediately "diving" into the activity of modeling the system in SysML.

Despite of its graphic syntax and its categorization among semi-formal languages, SysML indeed constrains the way of thinking systems in terms of diagrams. This is fully appropriate when the concern is to step-wise implement a system engineering process with its successive address of requirement capture, analysis and design steps. By contrast, elaborating SysML models first requires forms of brainstorming and sharing of knowledge (with clients, regulators). This step would benefit from a graphic support to express ideas, but in a way that is less formal and coercive than diagramming in SysML. The graphic support which is needed must be close to human way of addressing problems and thinking about systems.

This paper therefore addresses the issue of 'Thinking about the system before modeling it in SysML', and proposes to use mind maps for the $thinking$ part of the objective whilst keeping SysML as systems modeling language.

The paper is organized as follows. Section II presents the rationale behind the approach developed in the paper. Section III surveys related work. Section IV explains how the remainder of the paper sets mind maps upstream SysML v2 diagrams. Section V focuses discussion on real-time systems modeling. Sections VI, VII, VIII and IX respectively achieve needs expression, requirements analysis, use-case driven analysis, and design in a process associated with SysML. Section X concludes the paper and outlines future work.

II. Rationale

Developing a system, be it a product, a service, or an organisation, passes several life cycle phases. The first stage in a system's life cycle, the concept stage, focuses on understanding the implications of a system's mission and core functionality, a business case together with requirements, their interconnections and dependencies specified in *e.g.*, key performance indicators (KPIs) and trade-off indicators such as Figures of Merit (FoM) [4]. A threefold objective can be seen for the concept stage:

1) To interpret/understand a mission statement, supported by a positive (potential) business case,
2) To produce an initial definition of stakeholder requirements and KPIs with respect to the mission, and

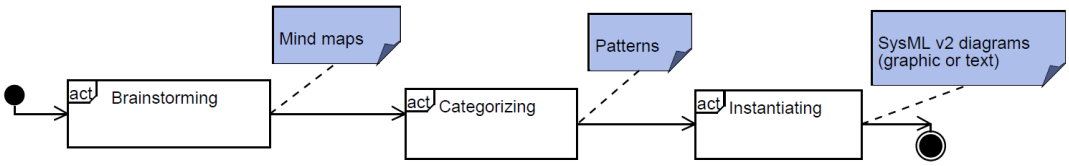

Fig. 1. The approach developed in the paper.

3) To produce an initial logical/conceptual description of a design.

The starting point for this phase concerns the needs and desires expressed by the stakeholders, which needs and desires are over the course of the concept phase transformed into requirements, potentially separated in Functional and Non-Functional Requirements, the latter potentially split in different types of non-functional requirements. This first phase, in which there may not be a very precise idea on what the stakeholders really want, is crucial as this is where the main decisions are taken. Once the requirements are considered clear and a first logical/conceptual description of the design is available, the main orientations are known. The developers then have a logical architecture of the system's design and its subsystems (the upper-level architecture) that meets system requirements: a preliminary design of the product, service or organisation to be developed.

The outcome of this first stage is orientating the next stages. It is important to provide developers with a good support to structure the information coming from the stakeholders, and to allow them to more clearly see the requirements space and the solution space. Starting from expressed needs and desires, from discussions, developers may find it complicated to formulate the requirements. This paper advocates the use of mind maps for this first phase and to have a transformation from mind maps to the first definition of requirements in a model-based systems engineering environment. Often used for structuring ideas or taking notes amongst others, a mind map is a diagram that allows for visually organise diverse types of information, as well as the relations between the different parts of the information [5] [6].

Mind maps can be seen as being built on top of semantic networks. Semantic networks are one of two closely related families of reasoning systems, the other family being description logics. Semantic networks date back to the early 20th century and offer graphical possibilities for visualisation of a knowledge base and efficient algorithms for inferring properties of an object on the basis of its category membership [7]. Closely related, description logics provide a formal language for constructing and combining category definitions and efficient algorithms for deciding subset and superset relationships between categories.

The intuitive use of mind maps to take notes and to structure knowledge, makes for a powerful tool. This paper investigates the possibilities of using mind maps as "front-end" to requirements engineering, so as to allow good, well-structured and well-supported collection of information of the concept stage. On the other hand, it also investigates a structured way to transform this information into requirements for further use.

III. Related Work

Mind maps assist persons in generating, classifying, categorizing and visualizing ideas. Many papers and books have addressed mind maps informally. In [8] Siochos and Papatheodorou formalize mind-maps as bi-partite graphs.

In [9] Quispe Vilchez and Pow-Sang Portillo survey literature on using mind maps in requirement engineering processes. Mind maps are identified as enablers of improved communication between the persons involved in the process of creating requirements and validating the latter.

Mind maps indeed need to be formalized to be used in conjunction with other modeling languages. For instance, to implement model transformation techniques. Such techniques are used in [10] where Wanderley, Belloir, Bruel, Hameurlain, and Araújo state that people in charge of eliciting requirements are not necessarily comfortable with SysML requirement diagrams. These authors propose to specify requirements using cognitive models such as mind maps. They use model transformation techniques to transform mind maps into KAOS (Knowledge Acquisition in Automated Specification) models that are subsequently transformed into SysML requirement diagrams. Thus, the paper by Wanderley *et al.* exclusively associates mind maps with requirement diagrams elicitation. Conversely, next sections of this paper consider that mind maps may influence not only requirement diagrams, but also use case, sequence and block diagrams. This paper also addresses needs, an important issue in systems engineering and an issue not addressed by any SysML diagram.

IV. Global Approach

For over a decade, the authors of this paper have been lecturing on SysML and supervising students projects. Students of various background have been trained, ranging from aerospace engineers to mechanical engineers, computer engineers, and building engineers. Despite of the broad variety of student profiles, the following issues may be addressed:

- `Experience in Systems Engineering` is often limited. Part of the students have been lectured on systems engineering, including MBSE, but the art of modeling systems is scarcely addressed in terms of what to do before developing a model. The idea of thinking

about the system before modeling is not so commonly addressed during SysML classes.

- `Making abstractions` is a stumbling block in the art of modeling and this is verified in the context of SysML. Newcomers to SysML need assistance. This paper proposes to use mind maps as a stepping stone to SysML diagrams creation.
- `Examples imitation` is a way of working that students use to feel comfortable. Instead of multiplying examples of closely similar systems, the concept of *pattern* enables modeling a reference for a class of systems. For instance, following sections of this paper are focused on the class of real-time systems.
- `Models are not just documentation artifacts` and need to be associated with tools and methods that enable addressing key issues, such as requirement traceability and checking of models against design errors. Therefore, next sections of this paper use SysML v2 and develop models with free software TTool [11].

Previous concerns lead the authors of this paper to develop the 3-step approach depicted by Fig.1 in the form of a SysML activity diagram. Three main activities are identified:

1) *Brainstorming* where the mind maps are created.
2) *Categorizing* where SysML diagram patterns are created.
3) *Instantiating* where the patterns are instantiated on one or several systems.

V. Software Controllers of Real-Time Systems

The term "Real-Time Systems" was coined to identify a class of systems that interact with their environment, run under the stimuli of the latter, and need to meet temporal constraints. There exists a large variety of real-time systems, from the simplest to the most sophisticated ones, ranging from microwave ovens to alarm controllers embedded onboard aircraft.

As far as modeling of real-time systems is concerned, the perimeter of the system needs to be clarified first. The problem may be phrased by a question: are we going to model the complete real-time system or a subset of it? That subset may be a (presumably software) controller that monitors the real-time system. In this paper, a real-time system controller was selected to be modeled, not the complete real-time system.

It is common practice to address a real-time system controller that is connected to a set of sensors and a set of actuators. Sensors provide the controller with input values and signals. Actuators are triggered by the outputs of the controller. Fig. 2 depicts a mind map that accordingly contains three branches for the sensors, the control function itself, and the actuators. A branch is added for maintenance, which is a key issue in systems design. The sensors (resp. the actuators) are categorized depending on whether they interact, or not, with human beings, particularly customers and supervisors. Control functions include the set up and shutdown procedures to be used when switching the real-time system on and off, respectively. Distinction is further made between nominal and degraded behaviors.

VI. Needs Expression

Needs definition is a crucial step for successful system conceptual design. The main objective of this step is to converge to a common top-level description of the system, compatible with the perspectives of all the stakeholders. The term 'stakeholders' refer to people, organizations and other systems having any interest into the system. The stakeholders include users, providers, maintenance team, authorities, and assurance companies [12]. The system description should be understandable by all people who are involved in its development, and will serve as a basis for planning the tasks and actions to be carried out.

SysML [2] does not offer any need definition diagram. Nor does SysML v2 [13]. In this paper, it is proposed to formulate needs in the form of sentences. The latter are labeled by identifiers.

Table I identifies needs associated with nominal behavior of the drink machine controller. Degraded behaviors - for instance "What to do when the drink machine fails connecting to the payment device?" - should also be taken into account. For space reasons, degraded behaviors are not addressed in this paper.

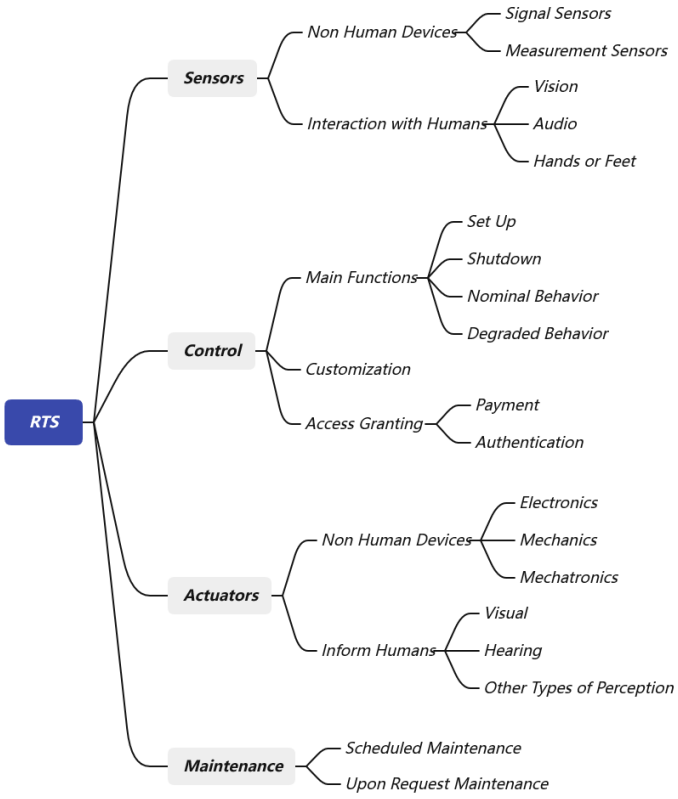

Fig. 2. Mind Map for Real-Time Systems.

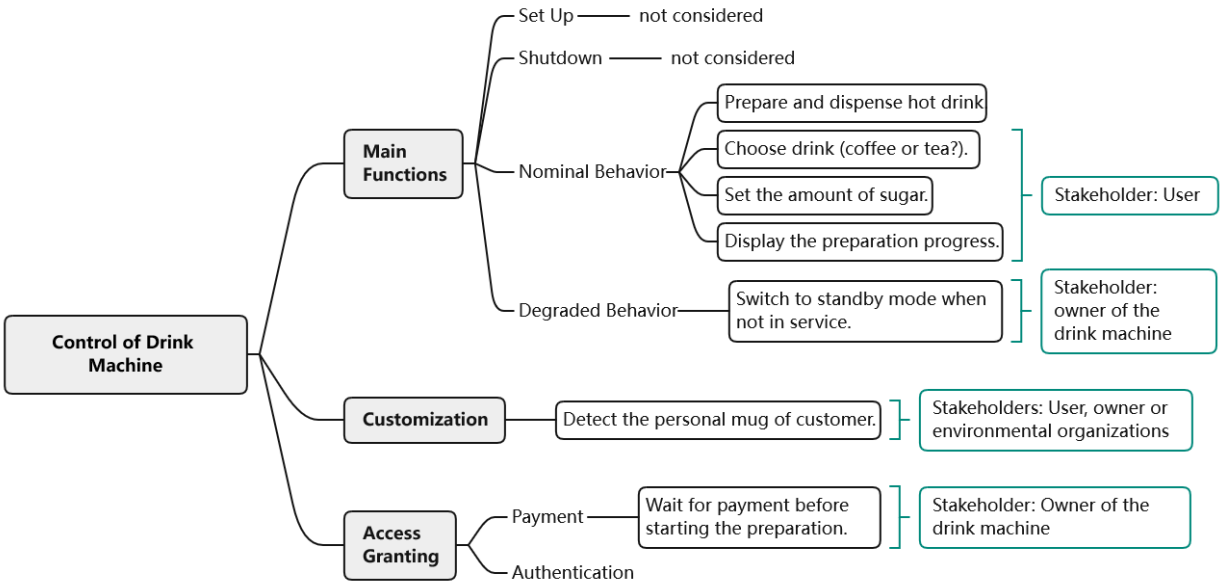

Fig. 3. Instantiating the Mind Map for Real-Time Systems : Drink machine.

TABLE I
PATTERN AND INSTANTIATION OF NEEDS DEFINITION FOR REAL-TIME SYSTEMS CONTROLLER

Pattern item		**Drink Machine**		
ID	Need	ID	Need	Priority
1	`Functional needs`: perform the functional needs for the system.	1.1	Mix ingredients to provide hot drinks.	H
		1.2	Choose drink (coffee or tea?).	H
		1.3	Set the amount of sugar.	M
		1.4	Detect customer's personal mug.	M
		1.5	Display the preparation progress.	L
		1.6	Switch to standby mode when not in service.	M
2	`Non-functional needs`: satisfy the non-functional needs for the system.	2.1	Manage the resources.	H
		2.2	Use electric power.	H
		2.3	Be able to clean the machine.	H
		2.4	Wait for payment before starting the preparation.	H

Table I contains the needs pattern which is instantiated on a drink machine controller. The pattern categorizes the needs in two groups:

1) `Functional needs`: they correspond to the functions that are the services offered by the system. For instance, the need 1.1 corresponding to *mixing the ingredients to provide hot drinks* is a function provided by the drink machine. This function instantiates the item `Non Human Devices` from the branch `Actuators` presented in the mind map in Fig. 2.
2) `Non-functional needs`: they define system attributes such as security, reliability, performance, maintainability, scalability, and usability, and serve as constraints or restrictions on a system design. For example, the need identified with the identifier 2.4 (*wait for payment before starting the preparation*) is derived from the item `Access Granting` of the branch `Control` as shown on Fig. 2.

Note that the same process associating needs (Tab. I) and mind map (Fig. 2) has been applied in this paper for writing the needs and the requirements.

Once needs have been written and specified in cooperation with the involved stakeholders, they should be prioritized. This paper reuses the three-level scale presented in [14]:

- $High$ (H): what is critical to complete the mission.
- $Medium$ (M): what is eventually required to complete the mission.
- Low (L): what would be nice to have someday if resources permit.

These needs are meant to be transformed into requirements. Defining and classifying them is very helpful for the next steps of the system development.

VII. Requirements Capture

Requirements have to be well-defined and specified to ensure successful development of new systems. To do so, collaborative work between the different stakeholders is necessary to translate their needs into requirements.

In this paper, the mind map depicted by Fig. 2 is used to guide the systems engineer when defining the requirements,

TABLE II
PATTERN FOR REQUIREMENTS DEFINITION FOR REAL-TIME SYSTEMS CONTROLLER

Pattern item	**Requirement attributes**			
	ID	Name	Text	Type (Sec. VII)
Sensors	S	Input	The system shall receive inputs.	F
	$S.1$	NoInteractionWithHumans	The system shall use sensors not interacting with Humans.	S
	$S.2$	InteractionWithHumans	The system shall use sensors interacting with Humans.	S
Control	C	Controller	The system shall process the inputs from the sensors.	F
	$C.1$	MainFunctions	The system shall perform its main functions.	F
	$C.2$	Customization	The system shall allow customization of the mission.	F
	$C.3$	AccessGranting	The system shall grant access to the user.	F
Actuators	A	Actuator	The system shall drive actuators.	F
	$A.1$	NonHumanDevices	The system shall use non-human devices.	S
	$A.2$	InformHumans	The system shall use devices to inform Humans.	S
Maintenance	M	Maintenance	The system shall be maintainable.	F
	$M.1$	ScheduledMaintenance	The system shall process scheduled maintenance operations.	F
	$M.2$	UnscheduledMaintenance	The system shall process unscheduled maintenance operations.	F

especially for real-time systems. The requirements are derived from the needs defined in section VI:

- `Sensors` requirements provide the system with the necessary inputs to perform its mission.
- `Control` requirements are related to the controller: they process the inputs received from the `Sensors` part to provide the controller's outputs to the `Actuators` part.
- `Actuators` requirements are related to the actions to be performed according to the controller's outputs.
- `Maintenance` requirements are related to maintaining the well functioning of the system.

Once requirements have been defined and understood, they can be classified into Functional and Non-Functional requirements. This last category can be split into three sub-categories such as Behavioral, Structural and Experiential as presented by Brazier, Van Langen, Lukosch and Vingerhoeds in [4]:

- Functional (**F**) requirements state the functions provided by the system.
- Behavioral (**B**) requirements specify the desired behavior of the system together with key performance indicators.
- Structural (**S**) requirements specify the components of the system and their relationships.
- Experiential (**E**) requirements define the desired impact of the system in the real world with real people.

This classification of requirements into these four categories helps systems engineers to know which requirements to consider at any point of the development stage [15]. Functional requirements are considered in particular during the use-case driven analysis step (see section VIII); whereas behavioral and structural requirements are used in the design step (see section IX). Experiential requirements are specific since they can be translated into functional, behavioral and structural ones.

```
requirement def id 'S.1' MugDetector {
    doc /* The system shall use a mug detector. */
    subject drinkMachine : DrinkMachine
}

requirement def id 'S.2' DrinkSelectionButton {
    doc /* The system shall use a drink selection button. */
    subject drinkMachine : DrinkMachine;
    attibute drinkChoice : drinkChoice
}

requirement def id 'C.1' DrinkPreparation {
    doc /* The system shall manage the drink preparation. */
    subject drinkMachine : DrinkMachine
}

requirement def id 'C.2' SugarQuantity {
    doc /* The system shall manage the sugar quantity. */
    subject drinkMachine : DrinkMachine;
    attibute sugarQuantity : sugarQuantity;
}

requirement def id 'C.3' Payment {
    doc /* The system shall process the payment. */
    subject drinkMachine : DrinkMachine;
    attibute drinkCost : drinkCost
}
```

Fig. 4. Instantiating the requirement table pattern on the drink machine using SysML v2 textual notation.

Table II depicts a requirement table pattern for real-time systems monitored by software controllers. This table contains an excerpt of high-level requirements, which can be further specified into lower-level ones to better describe the system. For instance, the requirement with the identifier $A.2$: `'The system shall use devices to inform Humans'` can be refined into $A.2.1$: `'The system shall use visual devices to inform Humans'` (S) and $A.2.2$: `'The system shall use sound devices to inform Humans'` (S).

The requirement pattern in Table II is instantiated on the controller of the drink machine which is depicted by Fig. 4, following SysML v2 [13] syntax. Compared to the previous version of SysML, Version 2 offers textual notation to model concepts including requirements which is intended to improve the precision, expressiveness, and usability [13].

VIII. USE CASE DIAGRAM

A use case diagram identifies the main functions or services to be offered by the system. Fig. 5 depicts a use case diagram pattern for software controllers of real-time systems. The use case diagram names the system

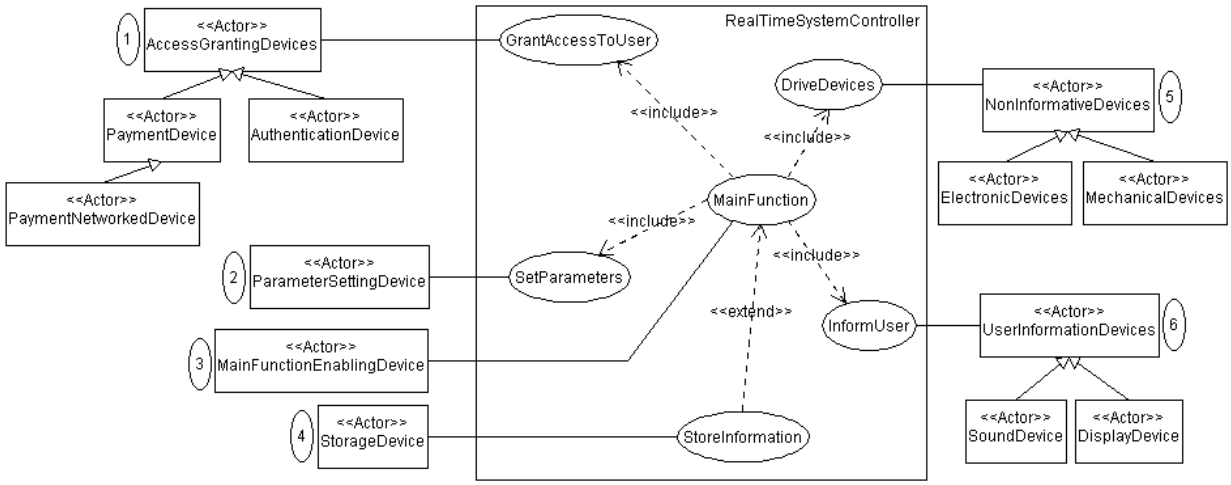

Fig. 5. Use Case Diagram: Pattern for Real-Time Systems Software Controller.

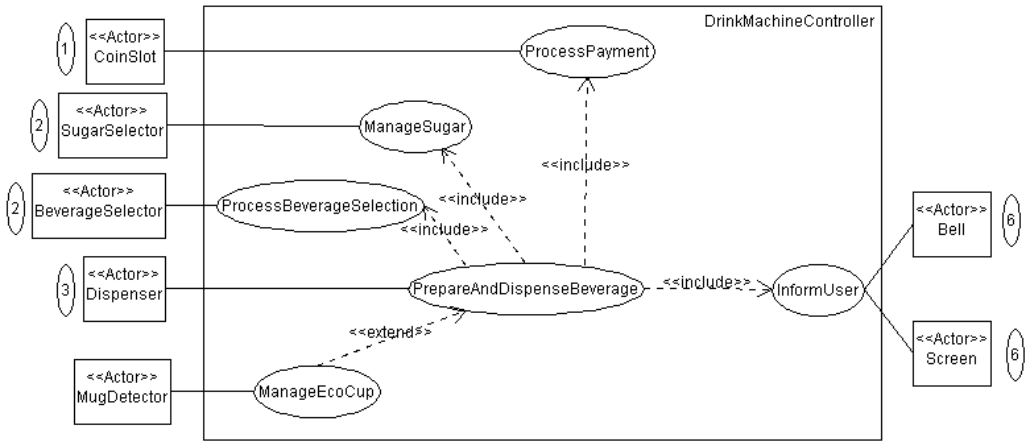

Fig. 6. Use Case Diagram for a Drink Machine.

as $Real\_Time\_System\_Controller$. A rectangle sets the boundary of the real-time system controller. The functions to be offered by the systems are identified by use cases drawn as ovals. The functions and the actors have been named relying on the branch labels of the mind map in Fig. 2.

Fig. 5 depicts the use case diagram associated with the $Sensors$, $Control$ and $Actuator$ branches of the mind map depicted by Fig. 2. For space reasons, the use case diagram associated with the $Maintenance$ branch of the same mind map is not shown in this paper.

The pattern depicted by Fig. 5 can now be instantiated. In this paper, the pattern is instantiated on a drink machine. It may be applied to other real time systems, such as microwave oven controllers and smartphones. The result of the instantiation is shown in Fig. 6.

Referring to Fig. 5, for instance, the use case `processBeverageSelection` is realized by the `BeverageSelector` actor that corresponds to the actor labelled by number 2 on Fig. 5 and Fig. 6. The same process is applied to the rest of the instantiation.

IX. Design step

The design step defines the architecture of the system in terms of interconnected blocks that SysML v2 names "parts" [13]. The latter distinguishes between part definition and part instantiation. Further, parts may be pairwise connected by $ports$.

Conforming to the approach depicted by Fig. 1, this section first proposes a diagram pattern and subsequently instantiates it on one concrete system, a drink machine controller.

Fig. 7 depicts a pattern for an architecture of real-time systems using SysML v2 textual notation, and referring to the mind map in Fig. 2. The maintenance part is not developed in this paper.

]

As shown in Fig. 7, each part is first defined in the section `'Definition of parts'` using the keyword `part def`. The part definition distinguishes be-

```
// DEFINITION OF PARTS
part def Block__RealTimeSystem {
    state entry {
        then exit;}
}
part def Block__RTS_Controller {
    in item in NHS_Ctrl_signal();
    out item out Ctrl_NHS_signal();
    in item in HS_Ctrl_signal();
    out item out Ctrl_HS_signal();
    in item in NHD_Ctrl_signal();
    out item out Ctrl_NHD_signal();
    in item in HD_Ctrl_signal();
    out item out Ctrl_HD_signal();
    state entry {
        then exit;}
}
part def Block__Sensors {
    state entry {
        then exit;}
}
part def Block__NonHumanInteraction_Sensors {
    out item out NHS_Ctrl_signal();
    in item in Ctrl_NHS_signal();
    state entry {
        then exit;}
}
part def Block__HumanInteraction_Sensors {
    out item out HS_Ctrl_signal();
    in item in Ctrl_HS_signal();
    state entry {
        then exit;}
}
part def Block__Actuators {
    state entry {
        then exit;}
}
part def Block__Human_Devices {
    out item out HD_Ctrl_signal();
    in item in Ctrl_HD_signal();
    state entry {
        then exit;}
}
part def Block__NonHumam_Devices {
    out item out NHD_Ctrl_signal();
    in item in Ctrl_NHD_signal();
    state entry {
        then exit;}
}
// USAGE OF PARTS
part RealTimeSystem :
Block__RealTimeSystem {
    part Actuators :
Block__Actuators;
    part RTS_Controller :
Block__RTS_Controller;
    part Sensors :
Block__Sensors;
}

// BINDING BETWEEN SIGNALS
bind (type=synchronous, isBroadcast=false, isPrivate=true, isLossy=false)NonHumanInteraction_Sensors.Ctrl_NHS_signal = RTS_Controller.Ctrl_NHS_signal;
bind (type=synchronous, isBroadcast=false, isPrivate=true, isLossy=false)NonHumanInteraction_Sensors.NHS_Ctrl_signal = RTS_Controller.NHS_Ctrl_signal;
bind (type=synchronous, isBroadcast=false, isPrivate=true, isLossy=false)HumanInteraction_Sensors.Ctrl_HS_signal = RTS_Controller.Ctrl_HS_signal;
bind (type=synchronous, isBroadcast=false, isPrivate=true, isLossy=false)HumanInteraction_Sensors.HS_Ctrl_signal = RTS_Controller.HS_Ctrl_signal;
bind (type=synchronous, isBroadcast=false, isPrivate=true, isLossy=false)RTS_Controller.HD_Ctrl_signal = NonHumam_Devices.NHD_Ctrl_signal;
bind (type=synchronous, isBroadcast=false, isPrivate=true, isLossy=false)RTS_Controller.Ctrl_HD_signal = NonHumam_Devices.Ctrl_NHD_signal;
bind (type=synchronous, isBroadcast=false, isPrivate=true, isLossy=false)RTS_Controller.NHD_Ctrl_signal = Human_Devices.HD_Ctrl_signal;
bind (type=synchronous, isBroadcast=false, isPrivate=true, isLossy=false)RTS_Controller.Ctrl_NHD_signal = Human_Devices.Ctrl_HD_signal;
```

Fig. 7. Architecture: Pattern for Real-Time Systems using SysML V2 textual notation.

```
// DEFINITION OF PARTS
part def Block__DrinkMachine {
    state entry {
        then exit;}
}
part def Block__Drink_Machine_Controller {
    private attribute SugarQuatity = 0 : int;
    out item out DetectMug();
    in item in MugDetected();
    in item in MugNotDetected();
    in item in CoinIsInserted();
    in item in SelectCoffee();
    in item in SelectTea();
    out item out PrepareCoffee();
    out item out PrepareTea();
    out item out AddSugar(int SugarQuantity);
    out item out EjectCup();
    in item in Completed();
    out item out Welcome();
    out item out HelpYourself();
    out item out RingBell();
    state entry {
        then exit;}
}
part def Block__Sensors {
    state entry {
        then exit;}
}
part def Block__Drink_Selector {
    out item out SelectCoffee();
    out item out SelectTea();
    state entry {
        then exit;}
}
part def Block__Mug_Detector {
    in item in DetectMug();
    out item out MugDetected();
    out item out MugNotDetected();
    state entry {
        then exit;}
}
part def Block__Coin_Slot {
    out item out CoinIsInserted();
    state entry {
        then exit;}
}
part def Block__Actuators {
    state entry {
        then exit;}
}
part def Block__Dispenser {
    private attribute SugarQuantity = 0 : int;
    in item in PrepareCoffee();
    in item in PrepareTea();
    in item in AddSugar(int SugarQuantity);
    in item in EjectCup();
    out item out Completed();
    state entry {
        then exit;}
}
part def Block__Bell {
    in item in RingBell();
    state entry {
        then exit;}
}
part def Block__Displayer {
    in item in Welcome();
    in item in HelpYourself();
    state entry {
        then exit;}
}
// USAGE OF PARTS
part DrinkMachine :
Block__DrinkMachine {
    part Sensors : Block__Sensors;
    part Actuators :
Block__Actuators;
    part Drink_Machine_Controller
:
Block__Drink_Machine_Controller;
}

// BINDING BETWEEN SIGNALS
bind (type=synchronous, isBroadcast=false, isPrivate=true, isLossy=false)Drink_Machine_Controller.Completed = Dispenser.Completed;
bind (type=synchronous, isBroadcast=false, isPrivate=true, isLossy=false)Drink_Machine_Controller.PrepareCoffee = Dispenser.PrepareCoffee;
bind (type=synchronous, isBroadcast=false, isPrivate=true, isLossy=false)Drink_Machine_Controller.PrepareTea = Dispenser.PrepareTea;
bind (type=synchronous, isBroadcast=false, isPrivate=true, isLossy=false)Drink_Machine_Controller.AddSugar = Dispenser.AddSugar;
bind (type=synchronous, isBroadcast=false, isPrivate=true, isLossy=false)Drink_Machine_Controller.EjectCup = Dispenser.EjectCup;
bind (type=synchronous, isBroadcast=false, isPrivate=true, isLossy=false)Drink_Machine_Controller.RingBell = Bell.RingBell;
bind (type=synchronous, isBroadcast=false, isPrivate=true, isLossy=false)Drink_Machine_Controller.Welcome = Displayer.Welcome;
bind (type=synchronous, isBroadcast=false, isPrivate=true, isLossy=false)Drink_Machine_Controller.HelpYourself = Displayer.HelpYourself;
bind (type=synchronous, isBroadcast=false, isPrivate=true, isLossy=false)Drink_Selector.SelectCoffee = Drink_Machine_Controller.SelectCoffee;
bind (type=synchronous, isBroadcast=false, isPrivate=true, isLossy=false)Drink_Selector.SelectTea = Drink_Machine_Controller.SelectTea;
bind (type=synchronous, isBroadcast=false, isPrivate=true, isLossy=false)Coin_Slot.CoinIsInserted = Drink_Machine_Controller.CoinIsInserted;
bind (type=synchronous, isBroadcast=false, isPrivate=true, isLossy=false)Mug_Detector.DetectMug = Drink_Machine_Controller.DetectMug;
bind (type=synchronous, isBroadcast=false, isPrivate=true, isLossy=false)Mug_Detector.MugDetected = Drink_Machine_Controller.MugDetected;
bind (type=synchronous, isBroadcast=false, isPrivate=true, isLossy=false)Mug_Detector.MugNotDetected = Drink_Machine_Controller.MugNotDetected
```

Fig. 8. Architecture of a drink machine using SysML v2 textual notation.

tween NonHumanInteraction_Sensors (NHS) and HumanInteraction_Sensors (HS) as far as Sensors are concerned. Similarly NonHuman_Devices (NHD) and Human_Devices (HD) are distinguished for the part Actuators. The four previously mentionned parts exchange signals with RTS_Controller. These signals are expressed by in item and out item for the input and output signals, respectively.

The section called 'Usage of parts' instantiates the part definition RealTimeSystem which shows that Sensors, Controller, and Actuators belong to the part RealTimeSystem.

One of the advantages of the textual notation of SysML v2 is that the links between two parts are visible in the last section called "Binding between signals". Indeed, the keyword bind describes the link between an input signal from the transmitter part and an output signal to the receiver part. For instance, in Fig. 9 the input signal Ctrl_NHS_signal is received by the part NonHumanInteraction_Sensors and it is associated with the output signal Ctrl_NHS_signal sent by the part RTS_Controller.

```
// Binding between signals
bind (type=synchronous, isBroadcast=false, isPrivate=true,
isLossy=false)NonHumanInteraction_Sensors.Ctrl_NHS_signal =
RTS_Controller.Ctrl_NHS_signal;
```

Fig. 9. Excerpt of Figure 7.

The RTS pattern is instantiated on a drink machine (Fig. 8). The drink machine's sensors part includes a Drink_Selector to select drinks, a Mug_Detector for sustainability concerns, and a Coin_Slot to proceed payments. These parts allow one to satisfy the needs 1.2 (Choose drink), 1.4 (Detect customer's personal mug) and 2.4 (Wait for payment before starting the preparation), respectively. The drink machine's actuators part includes a Dispenser which mix the ingredients to prepare hot drink, satisfying the need 1.1 (Mix ingredients to provide hot drinks).

The RTS pattern presented in this paper could be easily instantiated on different real time systems, such as electrically assisted bicycles, drones exploring buildings [16], and drones exploring high-voltage lines [17].

X. Conclusions

Whatever the modeling language, elaboration of models is a highly complex intellectual process and SysML is no exception. The authors of this paper propose to ease that elaboration process by using mind maps.

From their experience in teaching SysML modeling of systems, the authors of this paper may witness the interest of mind maps to help students thinking about the system to be modeled before developing SysML diagrams. Having one or several mind maps as a common reference to the requirement capture, analysis and design steps enables some kind of harmonization in the way requirement, modeling assumptions, use case, sequence and block diagrams are designed.

Unlike approaches that directly switch from mind maps to SysML diagrams dedicated to one specific system, this paper introduces an intermediate step: mind maps first enable elaboration of diagrams patterns. The latter may in turn be instantiated on one or several systems. The proposed approach is step-wise illustrated on real-time systems monitored by software controllers.

Further investigations are needed to partly automate the proposed approach inside a methodological assistant. The latter will advantageously combine Cased Based Reasoning Techniques with Mind Maps.

Acknowledgements

SysML diagrams and mind maps have been edited with free sofware TTool and Xmind, respectively. Thanks are due to Prof. Ludovic Apvrille for supporting SysMLv2 textual notation in TTool.

References

[1] *OMG Unified Modeling Language*, Object Management Group, https://www.omg.org/spec/UML/2.5/PDF, 2018.
[2] *OMG Systems Modeling Language*, Object Management Group, https://www.omg.org/spec/SysML/1.6, December 2019.
[3] S. Wolny, A. Mazak, C. Carpella, V. Geist, and M. Wimmer, "Thirteen years of SysML: A systematic mapping study," *Software and Systems Modeling*, vol. 19, p. 111–169, 2020.
[4] F. Brazier, P. v. Langen, S. Lukosch, and R. Vingerhoeds, *Complex Systems: Design, engineering, governance*, ser. Projects and People: Mastering Success. NAP Foundation Press, 2018.
[5] T. Buzan, *Mind Maps at Work: How to be the best at work and still have time to play*. Plume, 2005.
[6] V. Kokotovich, "Problem analysis and thinking tools: an empirical study of non-hierarchical mind mapping," *Design Studies*, vol. 29, no. 1, pp. 49–69, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0142694X07000889
[7] S. J. Russell and P. Norvig, *Artificial Intelligence: a modern approach*, 3rd ed. Pearson, 2016.
[8] V. Siochos and C. Papatheodorou, "Developing a formal model for mind maps," in *First Workshop on Digital Information Management*, 03 2011.
[9] E. Quispe Vilchez and J. A. Pow-Sang Portillo, "Mind maps in requirements engineering: A systematic mapping," in *Design, User Experience, and Usability. Design Philosophy and Theory*, A. Marcus and W. Wang, Eds. Cham: Springer International Publishing, 2019, pp. 335–350.
[10] F. Wanderley, N. Belloir, J.-M. Bruel, N. Hameurlain, and J. Araújo, "From goals to systems modeling: a user-centered modeling approach (in french)," *Inforsid*, pp. 113–128, 2014.
[11] TTool, "https://ttool.telecom-paris.fr/," *Retrieved September 10, 2021*, 2021.
[12] S. R. Hirshorn, L. D. Voss, and L. K. Bromley, *NASA Systems Engineering Handbook*, 2016.
[13] OMG, *SysML v2, https://www.omgsysml.org/SysML-2.htm*, 2021.
[14] K. Wiegers, "First things first: prioritizing requirements," *Software Development*, vol. 7, no. 9, pp. 48–53, 1999.
[15] J. J. M. Jiménez and R. Vingerhoeds, "A system engineering approach to predictive maintenance systems: from needs and desires to logical architecture," in *2019 International Symposium on Systems Engineering (ISSE)*. IEEE, 2019, pp. 1–8.
[16] E. Razafimahazo, P. de Saqui-Sannes, R. A. Vingerhoeds, C. Baron, J. Soulax, and R. Mège, "Mastering complexity for indoor inspection drone development," in *IEEE International Symposium on Systems Engineering, ISSE 2021, Vienna, Austria, September 13 - October 13, 2021*. IEEE, 2021, pp. 1–8.
[17] O. Aïello, D. S. D. R. Kandel, J. Chaudemar, O. Poitou, and P. de Saqui-Sannes, "Populating MBSE models from MDAO analysis," in *IEEE International Symposium on Systems Engineering, ISSE 2021, Vienna, Austria, September 13 - October 13, 2021*. IEEE, 2021, pp. 1–8.