An Effect Oriented Simulation Approch for Requirements Validation – Technical Documentation

I. Introduction

This is the supplementary document of paper "An Effect Oriented Simulation Approch for Requirements Validation". It provides the method and case study detailed descriptions.

The rest of the document is organised as follows. Section II introduces description and modeling basis. Section III defines how to write intentions and device scheduling scenarios. Section IV presents our framework, and Section V details our approach. Section VI presents a real-world case study and finally, Section VII concludes this paper, and puts forward future work.

II. Preliminaries

Throughout this paper, we will use a light controller problem as a running example. *This problem involves an operator and a light unit that can be turned on and off by receiving OnPulse and OffPulse signals*.

Based on Jackson's perspective [1], this paper adopts the fundamental concepts of the Problem Frames (PF) approach [2] to describe effects and scenarios. To capture the time-sensitive nature of device and environment effects, we utilize timed automata (TA) [3] as the behavior model and UPPAAL [4] as the simulation support platform. In the following, we provide an overview of these concepts.

A. Introduction to basic concepts in the PF

In the PF approach, the **software** environment is defined as a collection of **problem domains** that are directly or indirectly interactive. These domains, along with the software, are explicitly depicted in a problem diagram. An example of such a diagram is shown in Fig. 1. The *Light Controller (LC)* is the software controller that needs to be developed, represented by a rectangle with double vertical lines. The rectangles connected to the controller represent problem domains, which are categorized into three types: **causal, biddable, and lexical domains**. Causal domains (e.g., *Light Unit (LU)*, marked with C) demonstrate a predictable causal relationship between their phenomena. Biddable domains (e.g., *Operator (OP)*, marked with B) usually comprise of people, and the most significant feature is that they are physical, but there is no clear and predictable internal causality. The lexical domain represents the physical representation of data, marked with X.

The software will interact with *problem domains*, either directly (*Light Sensor (LS)* in Fig. 1) or indirectly (*Natural Environment (EN)* in Fig. 1). The interaction between the software and the problem domain is described by the syntax $sender!\{phenomenon[type]\}$. There are three types of phenomena: **event**, **state**, and **value**. For instance, $LC!\{onPulse[event]\}$ is an example of an interaction, which means that *LC sends event onPulse to LU*.

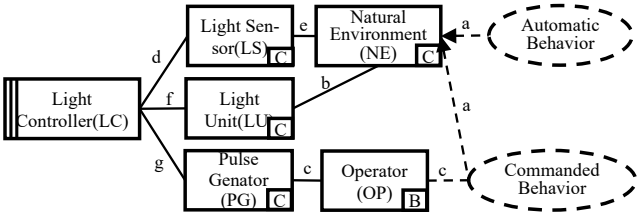


Fig. 1. A problem diagram of light control problem

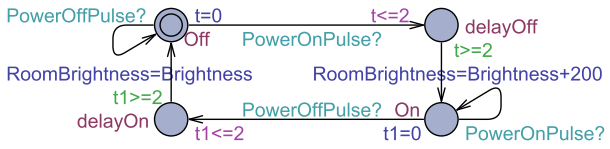
A requirement is a description of user needs, represented by a dotted ellipse, such as the *Commanded Behavior* shown in Fig. 1. Each requirement places constraints on, or refers to, expected changes in the problem domains. These constraints and references are represented by dotted lines and arrows in the PF. They take the form of interactions, where expected changes in the problem domains are depicted as phenomena. For instance, $LU!\{On[state]\}$ indicates that *LU* is expected to be turned *on*.

B. Introduction to Behavior model

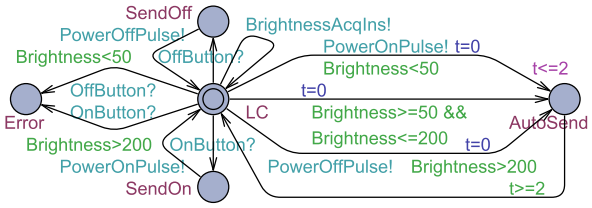
We plan to use TA to be the behavior model of software, devices and environment, and the network of TAs (NTA) to simulate the interactions between them. The definition of TA is as follows [3]. [Timed Automaton] A timed automaton is a tuple $(L, l_0, X, \sum, E, \mathcal{I})$ where: (i) L is a finite set of locations; (ii) $l_0 \in L$ is the initial location; (iii) X is a finite set of clocks; (iv) $\sum$ is a finite set of actions, i.e., synchronisations; (v) $E \subseteq L \times \sum \times \mathcal{G}(X) \times 2^X \times L$ is a finite set of edges with actions, guards, and a set of clocks to be reset, i.e., update, where $\mathcal{G}(X)$ is the set of guards over X; and (vi) $\mathcal{I} : L \to \mathcal{G}(X)$ assigns an invariant to each location.

We take *Light Unit TA* in Fig. 2(a) as an example to explain TA. There are 4 locations in *Light Unit* TA, i.e., *Off*, and *On*, and two other locations for expressing response time 2. In these two locations, there are two clock variables *t* and *t1*, and one invariant t_i*=2* to constrain the value of clock *t*, where *t* can increase over time. The transition from *off* to *delayOff* is triggered by a synchronization receiving (?) *PowerPulseOn*.

TAs communicate with each other through **shared variables** and **broadcast channels** to generate Networks of TAs



(a) Light Unit: TA



(b) Light Controller: TA

Fig. 2. Two TAs in the Light Controller problem

(NTA) [4]. The shared variables are declared at the system declaration and can be used in any TA in the NTA. The channels offer a way to ensure the synchronisation of the transitions in different TAs. Once the sending (!) is finished, the receiving (?) will be formed. For example, Fig. 2 gives two TAs of one NTA. The *Light Unit* TA shares variable *Brightness* with the *Light Controller* TA. It also has two synchronization channels with the *Light Controller* TA, *PowerOnPulse* and *PowerOffPulse*. The *Light Controller* TA sends (!) them while *Light Controller* TA receives (!) them.

III. Notations

This section firstly describes the effects and device scenarios. Then, we present our approach framework.

A. Description of effects

This paper differentiates between devices and environment based on problem domains. Devices are problem domains that directly interact with software and consist of sensors and actuators. They are considered causal domains. On the other hand, environment is a problem domain that interacts directly with devices. This can refer to a person whose instructions need to be followed, natural elements such as brightness and temperature, or controlled objects. People are considered biddable domains, while natural environment and controlled objects are considered causal domains.

We define effects to be a set of observable phenomena and their orders on environment. The phenomena are the same with that in the requirement references and requirement constraints. Based on that, we add orders. According to Mavin et al. 's Simple Requirements Syntax method (EARS) [5], [6], we choose one simple format "if $\cdots$ then " to express an effect.

For example, in Fig. 1, problem domains *Natural environment (NE)* and *Operator(OP)* are environment. According to the domain experts, effects includes:

EF1: *if brightness of natural environment is lower than 50 lux (higher than 200 lux), then adjust it more (less) bright*;

EF2: *if OP sends an OnButton(OffButton), then the brightness of natural environment will be higher (lower).*

Moreover, since the effects are time sensitive, we could also add time constraints on them. For example, EF2 could be:

if OP sends an OnButton (OffButton), then after 5s, the brightness is higher (lower).

B. Description of device scheduling scenarios

The device scheduling scenarios presented here refer specifically to scenarios based on software behavior and their relationships. As the software interacts with devices, it is essential to include interactive objects in the behavior descriptions.

Based on the PF approach, the behavior of the software is a result of its interactions with devices. Furthermore, after analyzing the requirements using the PF technique [2], [7], we have observed that these interactions sometimes involve lexical domains (data storage). In this paper, we also take into account the lexical domains as direct interaction objects, in addition to the devices.

Based on the interaction descriptions presented in the PF, we have defined two types of software behavior: "software sends(!) phenomena to causal (lexical domain)" and "software receives(?) phenomena from causal (lexical domain)". For instance, as demonstrated in Fig. 3, *LC* (the software) sends *BrightnessAcqIns* to *LS*. This behavior can be derived from the interactions presented in the problem diagram.

```
Automatic Behavior Scenario {
    while(true) {
        LC sends BrightnessAcqIns to LS[event];
        if(Brightness<50Lux) {
            LC sends PowerOnPulse to LU[event];
        }
        else if(Brightness>200Lux) {
            LC sends PowerOffPulse to LU[event];
        }
        after(5ms);
    }
}
```

Fig. 3. Automatic behavior scenario

In addition, scenarios encompass behavior relations that are specified using activity diagrams. Four types of behavior relations are identified: sequence (;), loop (while), parallel (——), and choice (if-else if-else) control. These are expressed using text format. Given that control is time-sensitive, two types of time constraints are added. The first constraint involves a specific time, such as at 5s do something, while the second constraint is the time delay between two behaviors, such as after 5ms. An example of a scenario for the light control problem is presented in Fig. 3, which specifies two choices for controlling the *light unit*.

IV. Our framework

Our approach simulates the behaviors of software, devices, and the environment to determine if the expected effects are achieved. We use TA as the behavior model to represent them individually, and employ both synchronization and shared data mechanism to simulate their interactions, resulting in an NTA. By analyzing simulation traces, we can validate whether the

expected effects are achieved. Our four-step framework for requirements validation is illustrated in Figure 4.

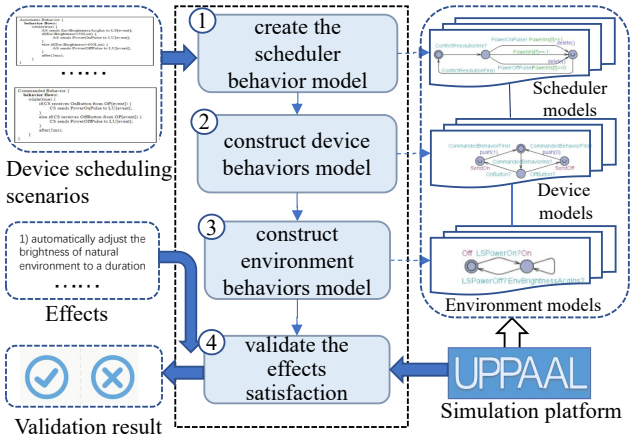


Fig. 4. Framework of our approach

Step 1: create the scheduler behavior model In this step, our goal is to develop scheduler behavior models based on device scheduling scenarios. Since these scenarios are interdependent, we propose generating a scheduler for each scenario to simulate a portion of the software's behavior. Each scheduler will directly interact with devices, and we must select an appropriate communication mechanism between them. The behaviors and their sequences specified in the scenarios will be converted into locations and transitions in the scheduler TA. This step will yield a set of scheduler TAs.

Step 2: construct the device behavior models This step involves constructing device TAs for all the devices involved in the scenarios. It's important to consider different device types, such as sensors and actuators, when constructing these TAs. It's also crucial to note that the devices not only interact with schedulers but also with the environment. Therefore, communication with scheduler TAs and the environment must be given due attention. Once this step is completed, a set of device TAs will be obtained.

Step 3: construct the environment behavior models The following step involves constructing environment TAs for all the environments involved in the scenarios. It is essential to consider the different types of environments, including person, natural environment, and controlled objects, while constructing these TAs. Since the environment interacts with devices, it is important to consider the communication mechanism with device TAs. Once this step is complete, we will have a set of environment TAs.

Step 4: validate the effects satisfaction In this step, we combine the scheduler TA, device TA, and environment TA to analyze the potential effects on the environment. The main challenge is to create an NTA that can be executed. This involves declaring models, channels, variables, and initializing them according to UPPAAL needs. By simulating the system, we can analyze the effects through simulation traces. Once all the desired effects have been achieved, we can confidently assert that the device scheduling scenarios are satisfactory.

V. Our Approach

In this section, we will provide a detailed explanation of each step in the framework. It's worth noting that communication mechanisms play a crucial role in every step of the process, and therefore, we will address this aspect at the outset.

A. Communication mechanism choice

In practice, the schedulers are responsible for scheduling devices, which may then be changed. These devices will interact with their environment to achieve certain effects. In other words, the effects are achieved through communication or collaboration. Based on the desired responses and characteristics of the devices, we can identify various communication mechanisms, as summarized in Table I.

Firstly, let's consider the communication between the scheduler and devices. We are dealing with two types of devices: sensors and actuators. Regardless of the kind of device or the phenomena they are sharing with the scheduler, we choose to implement a synchronization mechanism (one send!, one receive?) to ensure quick and efficient responses between the two. It's important to note that schedulers may interact with lexical domains, and so the communication mechanism we use is shared data, since lexical domains are for data storage.

Communicating between devices and the environment can be complex. The environment consists of biddable domains, such as operators, and causal domains, including the natural environment and satellites. Only sensors can sense the biddable domains, which should be quickly detected and responded to when they appear. To facilitate this, we continue to use synchronization mechanisms.

There are two situations that involve devices interacting with their causal environment. In some cases, causal environment entities simply share data with the devices. For example, the *light sensor* and the *natural environment* only share information about brightness, so shared data is used as the communication method. However, for controllable objects such as satellites, the devices need to send signals to them. In such cases, synchronization mechanisms are employed.

TABLE I
Communication Mechanisms Chosen

case	Participant I	Participant II	Phenomena type	Communication mechanism
1	scheduler	device	—	synchronization
2	scheduler	lexical	—	shared data
3	sensor	environment (B)	value,state	shared data
4	sensor	environment (B)	event	synchronization
5	device	environment (C)	value,state	shared data
6	device	environment (C)	event	synchronization

B. Scheduler behavior model creation

The scheduler will interact with various devices and must comply with the behavior flows in the device scheduling scenarios. These constraints will limit the capabilities of the scheduler TA. To ensure effective communication, we will follow the communication channels outlined in case 1 and 2

of Table I. As such, we will primarily construct the scheduler TAs based on behavior flows.

Within the behavior flows, there exist various behaviors that are interconnected. These behaviors serve to regulate the scheduler behaviors. When a behavior takes place, a specific location within the scheduler is designated for recording purposes. Therefore, we assign each behavior to a particular location and its connections to transitions. Additionally, every scenario must have an initial location. As scenarios will repeat themselves, there must be a transition back to the initial location to enable a fresh start.

In addition, we identify various scenario structures and their corresponding locations and transitions, as summarized in Figure 5. The software behaviors are represented by *b*, *b1*, and *b2*, while the previous location is represented by *Pre*.

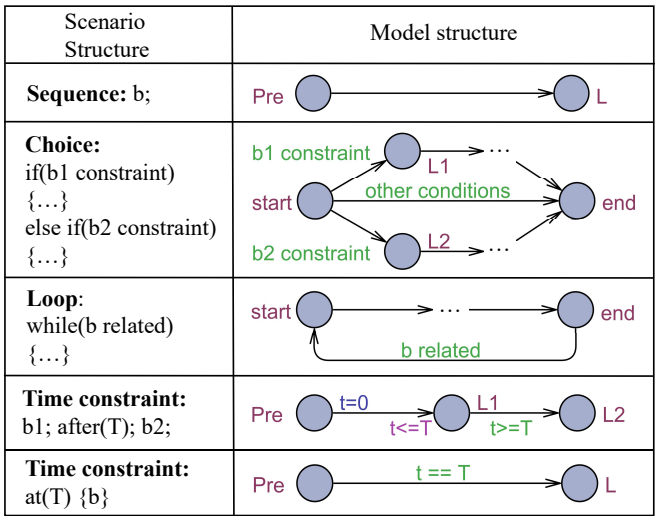


Fig. 5. TA model structures of different scenario structure

- For a sequence like "*b*;", add a transition directly from the previous location to its corresponding location *L*;
- For choice like "if (*b1 constraint*) {···} else if (*b2 constraint*) {···})", add 3 paths from the start node to end node, where 2 paths are from the two choices,*b1* constraint, and *b2* constraint, another one is for complementing the "else" conditions which will do nothing. It will directly transit to the end node. Moreover, the *b1* and *b2* constraints will be guard conditions in the transitions.
- For loop like "while (*b* related) {···}", there will be a start and an end node for the {···}, we need to add a transition from the end node to the start node using the b related as a guard.
- For time constraints like "*b1*; after(T); *b2*;", we reset the clock *t* to 0 before going to *L1* (corresponding to *b1*), and add invariant $t_{¡}=T$ in *L1*, and a guard $t_{¿}=T$ in the transition from *L1* to *L2*.
- For time constraints like "*at(T); b2;*", we declare a clock *t* at the initial location, and set a guard "*t=T*" in the transition from *Pre* location to *L*.

During the process, the behavior characteristic of *b* enhances the information in the transitions, as presented in Table II. Typically, devices that interact with schedulers are causal domains, and there are also lexical domains for saving data. Hence, we limit our consideration to only two scenarios:

1) If behavior *b* involves a lexical domain where communication involves saving or loading data, we simply need to record whether the operation has been completed. Therefore, we add an update such as "*content(b)=1*" in the transition to indicate that the data operation in *b* has been executed.

2) If a device is involved in the behavior *b*, the type of phenomenon within *b* (*content(b)*) will determine the communication mechanism required. If the content is a value or state, it can only be expressed through a shared variable. We apply the same process to lexical domains. If the content is an event, the communication mechanism will be synchronized to respond to the event. We use "*content(b)Ins!*" to indicate that *b* is sent by the scheduler, and "*content(b)Ins?*" to indicate receiving information.

TABLE II
TRANSITION INFORMATION FROM BEHAVIOR CHARACTERISTICS

Participant	Type (b)	Transition information
Lexical Domain	—	Add update in the transition, "content(b) = 1", declare "int content(b)"
Causal Domain	event	Add sync in the transition, if sends then sync="content(b)Ins!", if receives then sync="content(b)Ins?", declare "chan content(b)Ins"
	value/state	Add update in the transition, "content(b) = 1", declare "int content(b)"

We provide an example of how to obtain a scheduler TA using the *Automatic Behavior scenario*. In this scenario, there is a choice structure with two alternative paths (*Brightness¡50* and *Brightness¿200*) as depicted in Fig. 3. We identify a third path, *50¡=Brightness¡=200*, that connects the initial location to the delay location, as shown in Fig. 6. For the while structure, we include a transition from the delay location back to the initial location. Additionally, we introduce a time constraint of 5 seconds and add a guard $t_{¿}=5$ to the transition, while also adding an invariant $t_{¡}=5$ at the location delay.

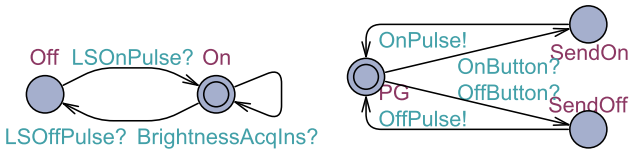
Fig. 6. Automatic Behavior scheduler TA

C. Device behavior model construction

In constructing the device behavior model, we adhere to the communication mechanism choices outlined in Case 1 and Cases 3 to 6 as presented in Table I. These mechanisms support a combination of shared data and synchronization. In

this section, we explore how to construct behavior models for various types of devices, including sensors and actuators.

Sensors are utilized to periodically monitor external environment variables or detect human behaviors. To construct the sensor TA that monitors attributes, we employ shared data as a communication mechanism. For instance, in Fig. 7, the *Light Sensor* shares the brightness attribute with the natural environment. In addition to communication mechanisms, we also consider work states and possible work state transitions. We map each work state into a location in TA. Typically, sensors have two work states, *off* and *on*. They can be time-driven or event-driven, with a self-transition updating the environment variables they monitor. When the scheduler acquires them, we add a synchronization message that is received from the scheduler. For example, in Fig. 7(a), the *Light Sensor* acquires the *environment brightness* by triggering the command *BrightnessAcqIns*.



(a) LightSensor (b) PulseGenerator

Fig. 7. Light Controller TAs: sensors

To develop the TA sensor for detecting human behaviors, the process of mapping from scenarios to locations and transitions is quite similar. The only variation lies in the communication synchronization that involves receiving human commands. For instance, as depicted in Fig. 7(b), the device *PulseGenerator* receives (?) the *OnButton* command from the *Operator* and subsequently sends (!) the *OnPulse* command to the scheduler.

Actuators are typically triggered by control signals and operate in a state-driven manner. These states are clearly defined, such as being either *on* or *off*. In TA modeling, actuators can be directly mapped to locations, with state transitions triggered by mapped transitions. As an example, a *Light Unit* can have two working states, *on* and *off*, which correspond to two distinct locations in the *TA* model. The state transition from *off* to *on* in Fig. 2(a) is triggered by the *"PowerOnPulse"* channel.

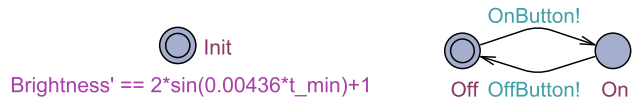
Some of the actuators, such as the *Light Unit* shown in Fig. 2(a), can affect environmental attributes, such as *brightness*. As a result, these actuators must interact with the environment through shared data. Additionally, different states of a device can impose various constraints on natural environmental attributes. These constraints are expressed as invariants, such as *Brightness'==1* in Fig. 2(a) when the *Light Unit* is in the "on" state, and *Brightness'==0* when the *Ligh tUnit* is "off". Actuators usually have a response time, and we can add pending locations after they receive signals but before they reach their expected states. A time invariant, such as t_i=2 in Fig. 2(a), can be used to specify this behavior.

D. Environment behavior model construction

Although there are three types of problem domains in the PF, we only have two types of environments - causal and biddable domains. Additionally, causal domains can be subcategorized into two types: natural environment and controllable objects. The natural environment, such as brightness, can be monitored and altered by devices. The controllable objects are similar to actuators, which receive signals to be controlled. The construction of controllable objects is similar to that of actuators, so we won't go into detail about it.

In order to construct the TA for the natural environment, we must consider its various attributes. Some of these attributes, such as room temperature, follow a specific natural law and continuously fluctuate. These attributes will be declared as invariants in a specific location, as illustrated in Fig. 8(a). Additionally, we must also acknowledge that certain attributes are not always continuous, such as whether it is raining or not. In these cases, we consider them as states and simulate the process of transitioning from rain to no rain over time.

When developing TA for a biddable domain involving human interaction, we focus solely on valid commands and their associated relations. Communication occurs through synchronization, with each valid command transmission or reception being associated with a specific location. Command sequences are then represented as transitions between these locations. As an example, in Fig. 8(b), the *Operator* sends (!) *OnButton* and *OffButton* commands alternatively.



(a) Natural environment (b) Operator

Fig. 8. Light Controller TAs: Environment model

E. Effects validation

Validation of the effects is checked through simulation traces, which are conducted using the UPPAAL platform. Prior to simulation, it is necessary to declare the necessary components for the NTA to be executed successfully.

(1) The model and broadcast channel declaration: The simulation system comprises various models, such as scheduler models, device models, and environment models. Channels are used for synchronizing these models and can be derived from the scheduler TAs, device TAs, and environment TAs.

(2)The declaration of global variables. These variables are shared between different TAs and include the state identifier of the device TA as well as the environment attributes in the environment TA. We have set the state identifiers of devices as integer variables as they represent the state of the device. Similarly, we have set the environment attributes as clock variables, such as *clock Brightness*, as we consider them to be continuously changing.

(3) The set of initial values The initial values of the variables are set differently. Specifically, we set the state identifier of

device TA to be the initial state of the device. As for the attributes of the environment, we can set them according to real-life cases.

After obtaining the system models for simulation using NTA, we use the UPPAAL platform to simulate them. We then execute the query as follows:

$simulate[<=t]\{a_1, a_2 \cdots, a_i, d_1, d_2 \cdots, d_n\}$

Where t is the simulation time; $a_1, a_2, \cdots, a_i$ are attributes to be monitored, e.g., *light brightness*; and $d_1, d_2, \cdots, d_n$ are the state variables of devices, such as *LightUnit*.

To determine whether the desired effects have been achieved, we can simply track the changes in the environmental attributes. For instance, when it comes to light control, the effects are limited to the operator and the brightness of natural environment. We can retrieve the necessary values and visualize the effects, as illustrated in Fig. 9.

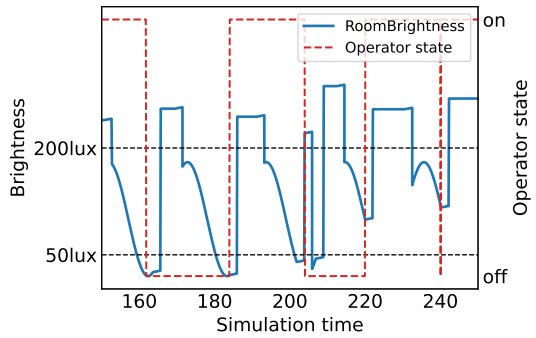


Fig. 9. Effects diagram

Based on the diagram, it is evident that the brightness increases whenever the *Operator* presses the *OnButton* and decreases when vice versa. Moreover, if the *brightness* exceeds 200 lux (or falls below 50 lux), it will automatically decrease (or increase) regardless of the Operator's actions. As a result, we can conclude that the anticipated outcomes outlined in EF1 and EF2 have been successfully attained.

Furthermore, we can also examine device variable changes to understand how the effects are achieved. By investigating the *light unit* and *light sensor*, we can uncover the process. For instance, when the *light sensor* detects that the *brightness* is less than 500 lux, the *light unit* will turn on, resulting in increased *brightness*.

VI. Case Study

We use a real world case study to show the feasibility of our approach. The case is about a spacecraft sun search problem. The sun search system needs to sense the position of the sun and the current attitude of its own satellites, and change its own attitude to achieve cruise to the sun. The problem statement is as follows: *The system is driven by a 32ms interrupt timer, collects data from the gyroscope, sun sensor and three-axis control thruster, receives ground commands through the data management computer, and automatically controls the thruster jet, so that the satellite rotates towards the sun.*

The sun search system has two tasks: *automatic sun tracking* and *responding to operator instruction*. The automatic sun tracking means is to collect data from sun sensors and gyros, and control the satellite attitude through the triaxial control thruster. Corresponding to operator's instruction is to receive instructions through the data management computer, and control the satellite according to the instructions. Fig. 10 shows its problem diagram. There are 4 devices, gyro, sun sensor, triaxial control thruster, and data management computer to interact with the Sun search controller. They will have influence on 3 environment entities, i.e., sun, satellite and ground operator. According to the domain experts, the expected effects are as follows:

1) *If the sun is visible, the satellite gets into CSM mode. If the sun is invisible, the satellite performs RDSM mode, PASM mode and RASM mode several times to search for the sun.*[1]

2) *When the ground operator sends CSM mode command or RDSM mode command or PASM mode command or RASM mode command, the satellite gets into correspondent mode respectively.*

We use the requirements projection method [2], [7] to decompose the sun search system. Finally we get 19 device scheduling scenarios. In these scenarios, there are 18 sequence structure, 1 choice structure, and 5 with time constraints. The details of each scenario are as follows.

A. Gyro Data Acquisition Scenario

Fig. 11(a) shows cases of a sequence structure with a time constraint that pertains to the Gyro Data Acquisition scenario. This scenario involves data collection from the *GY* (causal domain), followed by the storage of data to *GD* (lexical domain) after a 5ms delay.

Fig. 11(b) shows the Gyro Data Acquisition TA. This TA communicates with *GY* through the channels "Pulsecountacqlns" and "GyropowerstateperIns" and with the lexical domain *GD* through the shared data. And in the location Acq1 has a time delay.

B. Sun Sensor Data Acquisition Scenario

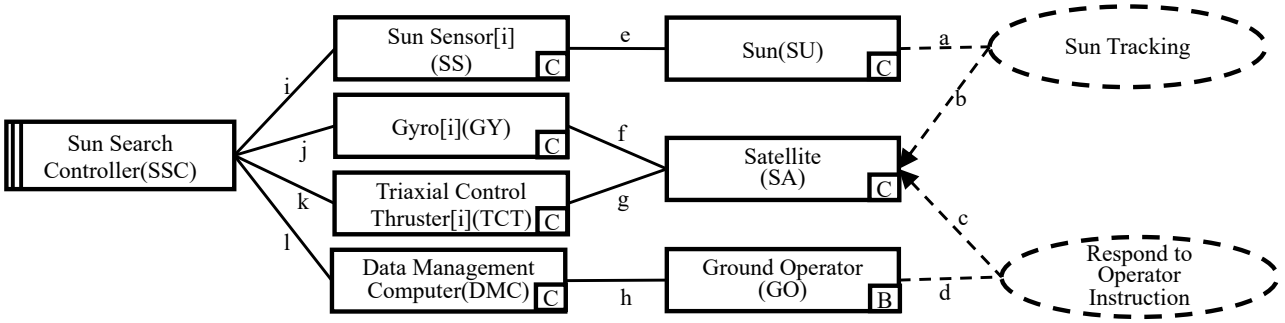
Fig. 12(a) shows cases a sequence structure with a time constraint that pertains to the Sun Sensor Data Acquisition scenario. This scenario involves data collection from the *SS* (causal domain), followed by the storage of data to *SSD* (lexical domain) after a 2ms delay.

Fig. 12(b) shows the Sun Sensor Data Acquisition TA. This TA communicates with *SS* through the channels "SunvisiblesignacqIns", "AngAnaAcqIns", and "Sunsensorpowerstateperlns!", and with the lexical domain *SSD* through the shared data. And in the location Acq1 and Acq2 have a time delay.

C. Thruster Data Acquisition Scenario

Fig. 13(a) shows cases of a sequence structure with a time constraint that pertains to the Thruster Data Acquisition scenario. This scenario involves data collection from the *TCT*

[1]By visible and invisible, it means the sun is at a certain angle. Due to the security needs, we hide them.

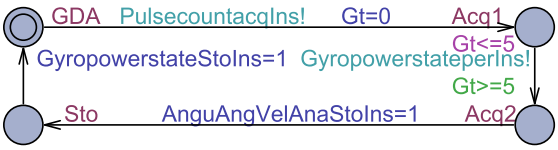


a: SSC!{Sense the position of the sun[event]} b: SSC!{Control satellite attitude[event]} c: SSC!{Set satellite mode[event]}
d: GO!{Command[event]} e: SS!{Sun visible information[value], Sun angle[value]} f: GY!{Angular velocity analog[value]}
g: TCT!{Controlling moment[signal]} h: GO!{RDSM command[event], CSM command[event]}
i: SSC!{Sun sensor power on pulse[event], Sun visible sign acquisition instruction[event], Angle analog acquisition instruction[event], Sun sensor power state perception instruction[event]}
j: SSC!{Gyro power on pulse[event], Pulse count acquisition instruction[event], Gyro power state perception instruction[event]}
k: SSC!{Thruster power on pulse[event], Thruster power state perception instruction[event], Triaxial Control Signal[signal]}
l: DMC!{Telecontrol instruction[event]}, SSC!{Telemetry data transmission instruction[event]}

Fig. 10. Sun search system before decomposition - problem diagram

```
Gyro Data Acquisition Scenario{
    GDA sends PulseCountAcqIns to GY[event];
    after(5ms);
    GDA sends GyroPowerStatePerIns to GY[event];
    GDA sends AngVelAnaStoIns to GD[event];
    GDA sends GyroPowerStateStoIns to GD[event];
}
```

(a) Gyro Data Acquisition Scenario



(b) Gyro Data Acquisition TA

Fig. 11. Gyro Data Acquisition

(causal domain), followed by the storage of data to *TD* (lexical domain) after a 5ms delay.

Fig. 13(b) shows the Thruster Data Acquisition TA. This TA communicates with *TCT* through the channel "Thruster-powerstateperIns", and with the lexical domain *TD* through the shared data. And in the location Acq1 has a time delay.

D. 32ms Interrupt Timer Initialization Scenario

Fig. 14(a) shows cases of a sequence structure that starts the 32ms Interrupt Timer scenario. This scenario involves sending start instruction to *32IT*(causal domain).

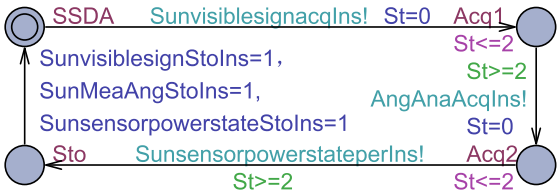
Fig. 14(b) shows the 32ms Interrupt Timer Initialization TA. This TA communicates with *32IT* through the channel "ITStaIns". And in the location Receive has a time delay.

E. Store Gyro Power on Instruction Scenario

Fig. 15(a) shows cases of a sequence structure that saves the Gyro Power on Instruction scenario. This scenario involves

```
Sun Sensor Data Acquisition Scenario{
    SSDA sends SunVisibleSignAcqIns to SS[event];
    after(2ms);
    SSDA sends AngAnaAcqIns to SS[event];
    after(2ms);
    SSDA sends SunSensorPowerStatePerIns to
SS[event];
    SSDA sends SunVisibleSignStoIns to SSD[event];
    SSDA sends SunMeaAngStoIns to SSD[event];
    SSDA sends SunSensorPowerStateStoIns to
SSD[event];
}
```

(a) Sun Sensor Data Acquisition Scenario



(b) Sun Sensor Acquisition TA

Fig. 12. Sun Sensor Data Acquisition

sending Gyro Power on Instruction Storage Instruction to *GI*(lexical domain).

Fig. 15(b) shows the Store Gyro Power on Instruction TA. This TA communicates with the lexical domain *GI* through the shared data "GyropowerIns". And we use the data "GyropowerInsStoIns=1" to denote that the data is stored in the lexical domain *GI*.

F. Store Sun Sensor Power on Instruction Scenario

Fig. 16(a) shows cases of a sequence structure that saves the Sun Sensor Power on Instruction scenario. This scenario

```
Thruster Data Acquisition Scenario{
    TDA sends ThrusterPowerStatePerIns to
TCT[event];
    after(5ms);
    TDA sends ThrusterPowerStateStoIns to
TD[event];
}
```

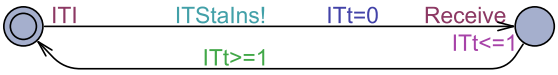
(a) Thruster Data Acquisition Scenario

(b) Thruster Data Acquisition TA

Fig. 13. Thruster Data Acquisition

```
32ms Interrupt Timer Initialization Scenario{
    32ITI sends 32ITStaIns to 32IT[event];
    after(1ms);
}
```

(a) 32ms Interrupt Timer Initialization Scenario



(b) 32ms Interrupt Timer Initialization TA

Fig. 14. 32ms Interrupt Timer Initialization

involves sending Sun Sensor Power on Instruction Storage Instruction to *SSI*(lexical domain).

Fig. 16(b) shows the Store Sun Sensor Power on Instruction TA. This TA communicates with the lexical domain *SSI* through the shared data "SunsensorpowerIns". And we use the data "SunsensorpowerInsStoIns=1" to denote that the data is stored in the lexical domain *SSI*.

G. Store Thruster Power on Instruction Scenario

Fig. 17(a) shows cases of a sequence structure that saves the Thruster Power on Instruction scenario. This scenario involves sending Thruster Power on Instruction Storage Instruction to *TPI*(lexical domain).

Fig. 17(b) shows the Store Thruster Power on Instruction TA. This TA communicates with the lexical domain *TPI*

```
Store Gyro Power on Instruction Scenario{
    SGPonI sends GyroPowerOnInsStoIns to
GI[event];
}
```

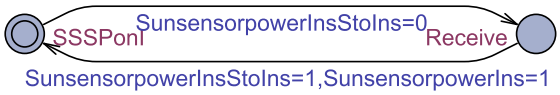
(a) Store Gyro Power on Instruction Scenario

(b) Store Gyro Power on Instruction TA

Fig. 15. Store Gyro Power on Instruction

```
Store Sun Sensor Power on Instruction Scenario{
    SSSPonI sends SunSensorPowerOnInsStoIns to
SSI[event];
}
```

(a) Store Sun Sensor Power on Instruction Scenario



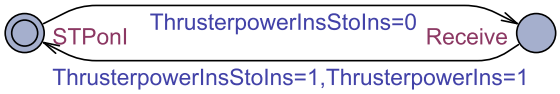
(b) Store Sun Sensor Power on Instruction TA

Fig. 16. Store Sun Sensor Power on Instruction

through the shared data "ThrusterpowerIns". And we use the data "ThrusterpowerInsStoIns=1" to denote that the data is stored in the lexical domain *TPI*.

```
Store Thruster Power on Instruction Scenario{
    STPonI sends ThrusterPowerOnInsStoIns to
TPI[event];
}
```

(a) Store Thruster Power on Instruction Scenario



(b) Store Thruster Power on Instruction TA

Fig. 17. Store Thruster Power on Instruction

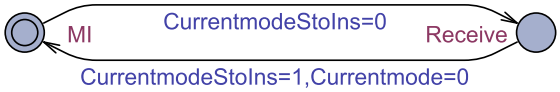
H. Mode Initialization Scenario

Fig. 18(a) shows cases of a sequence structure that saves the Current Mode scenario. This scenario involves sending Current Mode Storage Instruction to *MR*(lexical domain).

Fig. 18(b) shows the Mode Initialization TA. This TA communicates with the lexical domain *MR* through the shared data "Currentmode". And we use the data "CurrentmodeStoIns=1" to denote that the data is stored in the lexical domain *MR*.

```
Mode Initialization Scenario{
    MI sends CurModeStoIns to MR[event];
}
```

(a) Mode Initialization Scenario



(b) Mode Initialization TA

Fig. 18. Mode Initialization

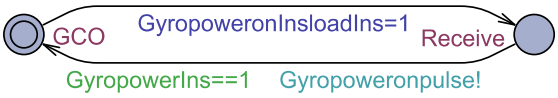
I. Gyro Control Output Scenario

Fig. 19(a) portrays a sequence structure known as the Gyro Control Output scenario. It power on *GY* (causal domain) based on data obtained from *GI*(lexical domain).

Fig. 19(b) shows the Gyro Control Output TA. This TA communicates with *GY* through the channel "Gyropoweronpulse", and with the lexical domain *GI* through the shared data "GyropowerIns". And we use the data "GyropoweronInsloadIns=1" to denote that the data is loaded from the lexical domain *GI*.

Gyro Control Output Scenario{
GCO sends GyroPowerOnInsLoadIns to GI[event];
GCO sends GyroPowerOnPulse to G[event];
}

(a) Gyro Control Output Scenario



(b) Gyro Control Output TA

Fig. 19. Gyro Control Output

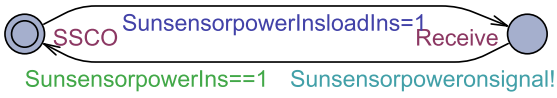
J. Sun Sensor Control Output Scenario

Fig. 20(a) portrays a sequence structure known as the Sun Sensor Control Output scenario. It power on *SS* (causal domain) based on data obtained from *SSI*(lexical domain).

Fig. 20(b) shows the Sun Sensor Control Output TA. This TA communicates with *SS* through the channel "Sunsensorpoweronsignal", and with the lexical domain *SSI* through the shared data "SunsensorpowerIns". And we use the data "SunsensorpowerInsloadIns=1" to denote that the data is loaded from the lexical domain *SSI*.

Sun Sensor Control Output Scenario{
SSCO sends SunSensorPowerOnInsLoadIns to SSI[event];
SSCO sends SunSensorPowerOnPulse to SS[event];
}

(a) Sun Sensor Control Output Scenario



(b) Sun Sensor Control Output TA

Fig. 20. Sun Sensor Control Output Scenario

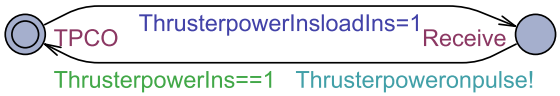
K. Thruster Power Control Output Scenario

Fig. 21(a) portrays a sequence structure known as the Thruster Power Control Output scenario. It power on *TCT* (causal domain) based on data obtained from *TPI*(lexical domain).

Fig. 21(b) shows the Thruster Power Control Output TA. This TA communicates with *TCT* through the channel "Thrusterpoweronpulse", and with the lexical domain *TPI* through the shared data "ThrusterpowerIns". And we use the data "ThrusterpowerInsloadIns=1" to denote that the data is loaded from the lexical domain *TPI*.

Thruster Power Control Output Scenario{
TPCO sends ThrusterPowerOnInsLoadIns to TPI[event];
TPCO sends ThrusterPowerOnPulse to TCT[event];
}

(a) Thruster Power Control Output Scenario



(b) Thruster Power Control Output TA

Fig. 21. Thruster Power Control Output

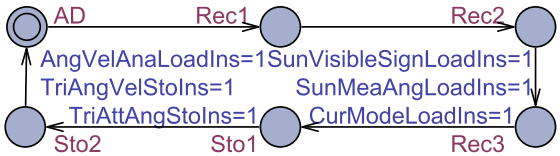
L. Attitude Determination Scenario

Fig. 22(a) depicts a sequence structure concerning the Attitude Determination scenario which obtains input data from *GD* (lexical domain), *SSD* (lexical domain) and *MR* (lexical domain), and stores output data to *ADR* (lexical domain).

Fig. 22(b) shows the Attitude Determination TA. This TA communicates with the lexical domain *GD*, *SSD*, *MR*, and *ADR* through the shared data. And we use the data "AngVelAnaLoadIns=1", "SunVisibleSignLoadIns=1", "SunMeaAngLoadIns=1", and "CurModeLoadIns=1" to denote that the data is loaded from the lexical domain *GD*, *SSD*, and *MR*. And we use the data "TriAttAngStoIns=1" and "TriANgVelStoIns=1" to denote that the data is stored in the lexical domain *ADR*.

Attitude Determination Scenario{
AD sends AngVelAnaLoadIns to GD[event];
AD sends SunVisibleSignLoadIns to SSD[event];
AD sends SunMeaAngLoadIns to SSD[event];
AD sends CurModeLoadIns to MR[event];
AD sends TriAttAngStoIns to ADR[event];
AD sends TriAngVelStoIns to ADR[instruction];
}

(a) Attitude Determination Scenario



(b) Attitude Determination TA

Fig. 22. Attitude Determination

M. Mode Switching Management Scenario

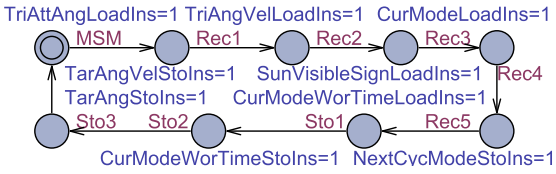
Fig. 23(a) depicts a sequence structure concerning the Mode Switching Management scenario which obtains input data from *ADR* (lexical domain), *SSD* (lexical domain) and

MR (lexical domain), and stores output data to *MR* (lexical domain).

Fig. 23(b) shows the Mode Switching Management TA. This TA communicates with the lexical domain *ADR*, *SSD*, and *MR* through the shared data. And we use the data "TriAttAngLoadIns=1", "TriAngVelLoadIns=1", "SunVisibleSignLoadIns=1", "CurModeLoadIns", and "CurModeWorTimeLoadIns=1" to denote that the data is loaded from the lexical domain *ADR*, *SSD*, and *MR*. And we use the data "NextCycModeStoIns", "CurModeWorTimeStoIns=1", "TarAngStoIns=1", and "TarAngVelStoIns=1" to denote that the data is stored in the lexical domain *MR*.

```
Mode Switching Management Scenario{
    MSM sends TriAttAngLoadIns to ADR[event];
    MSM sends TriAngVelLoadIns to ADR[event];
    MSM sends SunVisibleSignLoadIns to SSD[event];
    MSM sends CurModeLoadIns to MR[event];
    MSM sends CurModeWorTimeLoadIns to
MR[event];
    MSM sends NextCycModeStoIns to MR[event];
    MSM sends CurModeWorTimeStoIns to
MR[event];
    MSM sends TarAngStoIns to MR[event];
    MSM sends TarAngVelStoIns to MR[event];
}
```

(a) Mode Switching Management Scenario



(b) Mode Switching Management TA

Fig. 23. Mode Switching Management

N. Thruster Control Computing Scenario

Fig. 24(a) depicts a sequence structure concerning the Thruster Control Computing scenario which obtains input data from *ADR* (lexical domain) and *MR* (lexical domain), and stores output data to *CCR* (lexical domain).

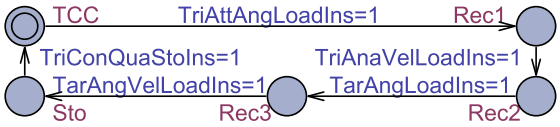
Fig. 24(b) shows the Thruster Control Computing TA. This TA communicates with the lexical domain *ADR* and *MR* through the shared data. And we use the data "TriAttAngLoadIns=1", "TriAnaVelLoadIns=1", "TarAngLoadIns=1", and "TarAngVelLoadIns=1" to denote that the data is loaded from the lexical domain *ADR*, and *MR*. And we use the data "TriConQuaStoIns=1" to denote that the data is stored in the lexical domain *CCR*.

O. Thruster Triaxial Control Scenario

Fig. 25(a) portrays a choice structure with a time constraint known as the Thruster Triaxial Control Output scenario. It counts the interrupt signal and when the count is equal to 5, saves the Triaxial Control Instruction to the *TTI*(lexical domain).

```
Thruster Control Computing Scenario{
    TCC sends TriAttAngLoadIns to ADR[event];
    TCC sends TriAnaVelLoadIns to ADR[event];
    TCC sends TarAngLoadIns to MR[event];
    TCC sends TarAngVelLoadIns to MR[event];
    TCC sends TriConQuaStoIns to CCR[event];
}
```

(a) Thruster Control Computing Scenario



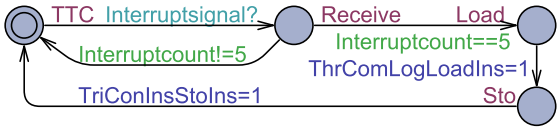
(b) Thruster Control Computing TA

Fig. 24. Thruster Control Computing

Fig. 25(b) shows the Thruster Triaxial Control TA. This TA communicates with the 32IT through the channel "Interruptsignal", and the lexical domain *TDCR* and *TTI* through the shared data. And there is a branch structure in the Receive location. The transition from Receive to Load is built based on the scenario, and then the transition from Receive to TTC is added as needed for the model to run correctly. And we use the data "ThrComLogLoadIns=1" to denote that the data is loaded from the lexical domain *TDCR*. And we use the data "TriConInsStoIns=1" to denote that the data is stored in the lexical domain *TTI*.

```
Thruster Triaxial Control Scenario{
    TTC receives Interrupt signal from 32IT[event];
    if (Interrupt count == 5) {
        TTC sends ThrComLogLoadIns to
TDCR[event];
        TTC sends TriConInsStoInst to TTI[event];
    }
    after(32ms);
}
```

(a) Thruster Triaxial Control Scenario



(b) Thruster Triaxial Control TA

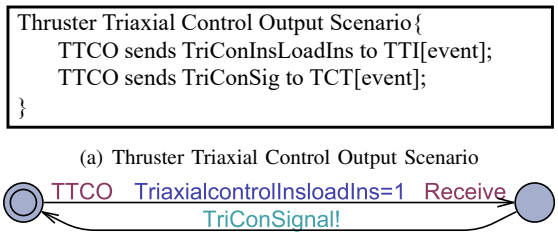
Fig. 25. Thruster Triaxial Control

P. Thruster Triaxial Control Output Scenario

Fig. 26(a) depicts a sequence structure concerning the Thruster Triaxial Control Output scenario which controls *TCT* (causal domain) based on data obtained from *TTI*(lexical domain).

Fig. 26(b) shows the Thruster Triaxial Control Output TA. This TA communicates with the TCT through the channel "TriConSignal", and with the lexical domain *TTI* through the

shared data. And we use the data "TriaxialcontrolInsload-Ins=1" to denote that the data is loaded from the lexical domain *TTI*.



(a) Thruster Triaxial Control Output Scenario

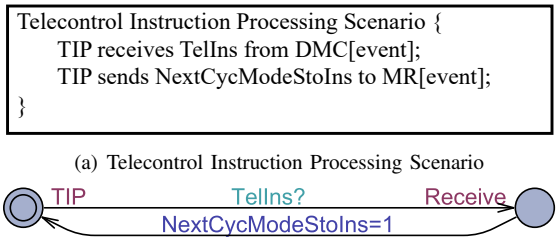
(b) Thruster Triaxial Control Output TA

Fig. 26. Thruster Triaxial Control Output

Q. Telecontrol Instruction Processing Scenario

Fig. 27(a) depicts a sequence structure concerning the Telecontrol Instruction Processing scenario which saves Next Cycle Mode based on Telecontrol Instruction obtained from *DMC*(causal domain).

Fig. 27(b) shows the Telecontrol Instruction Processing TA. This TA communicates with the DMC through the channel "TelIns", and with the lexical domain *MR* through the shared data. And we use the data "NextCycModeStoIns=1" to denote that the data is stored in the lexical domain *MR*.



(a) Telecontrol Instruction Processing Scenario

(b) Telecontrol Instruction Processing TA

Fig. 27. Telecontrol Instruction Processing

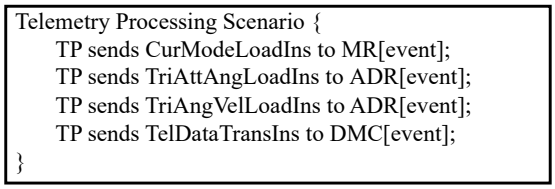
R. Telemetry Processing Scenario

Fig. 28(a) depicts a sequence structure concerning the Telemetry Processing scenario which packages data and sends them to *DMC*(causal domain).

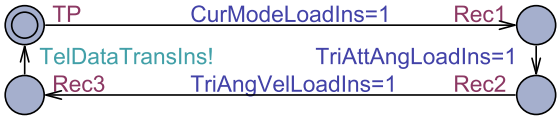
Fig. 28(b) shows the Telemetry Processing TA. This TA communicates with the DMC through the channel "TelData-TransIns", and with the lexical domain *MR* and *ADR* through the shared data. And we use the data "CurModeLoadIns=1", "TriAttAngLoadIns=1", and "TriAngVelLoadIns=1" to denote that the data is loaded in the lexical domain *MR* and *ADR*.

VII. Conclusion

This paper presents a novel approach for validating requirements by simulating the connections between effects and device scheduling scenarios. Our method involves describing environment effects, and then simulating interactions between schedulers, devices, and the environment to achieve those



(a) Telemetry Processing Scenario



(b) Telemetry Processing TA

Fig. 28. Telemetry Processing

effects. In contrast to conventional methods, our validation process meticulously accounts for the physical attributes of devices and the environment. To establish the viability of our approach, we conducted a case study in the spacecraft domain.

In this paper, we acknowledge the limitations of our environmental considerations, which have been confined to specific situations. We recognize that the environment is inherently uncertain and dynamic, and we aim to address this challenge in future research endeavors. To this end, we plan to conduct additional case studies and develop innovative tools that can facilitate the validation of requirements in complex and dynamic environments.

References

[1] J. M, "The meaning of requirements," *Ann. Softw. Eng.*, vol. 3, pp. 5–21, 1997.

[2] M. Jackson, *Problem frames: analysing and structuring software development problems*. Addison-Wesley, 2001.

[3] B. J and Y. W, "Timed automata: Semantics, algorithms and tools," in *Advanced Course on Petri Nets*, 2003, pp. 87–124.

[4] A. D, K. L, A. L, and et al, "Uppaal SMC tutorial," *Int. J. Softw. Tools Technol. Transf.*, pp. 397–415, 2015.

[5] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak, "Easy approach to requirements syntax (ears)," in *2009 17th IEEE International Requirements Engineering Conference*. IEEE, 2009, pp. 317–322.

[6] A. M. Mav and P. Wilkinson, "Ten years of ears," *IEEE Software*, vol. 36, no. 5, pp. 10–14, 2019.

[7] Z. Jin, X. Chen, and D. Zowghi, "Performing projection in problem frames using scenarios," in *APSEC*, 2009, pp. 249–256.