An Effect Oriented Simulation Approch for Requirements Validation – Technical Documentation

I. Introduction

This is the supplementary document of the paper "An Effect Oriented Simulation Approach for Requirements Validation". It provides the case study with detailed descriptions in Section II.

II. 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. 1 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 [1], [2] 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 this case study are as follows.

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

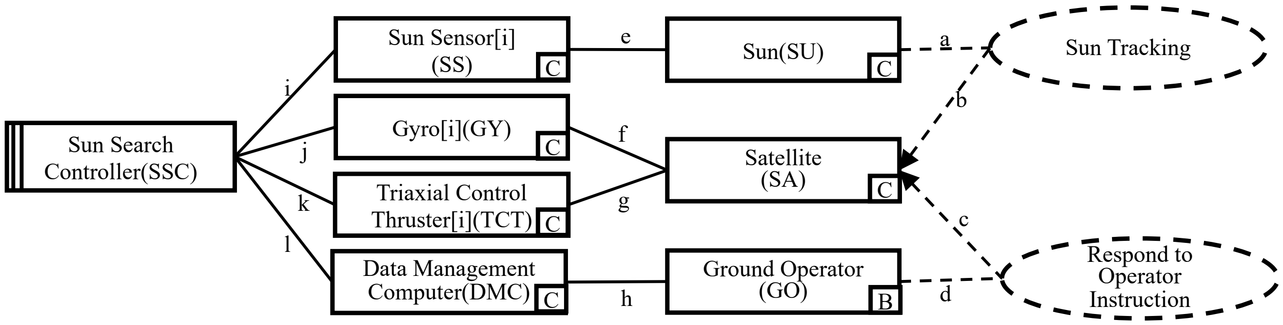
A. Device behavior modeling

In this case, there are three types of sensors: the *gyro*, the *sun sensor*, and the *data management computer*, as well as an actuator *triaxial control thruster*. The *gyro* is employed to measure the angles of the *Satellite* via "Pulsecount". Therefore, it interacts with both the scheduler and the causal domain *Satellite*. The *gyro* communicates with the *Satellite* through shared data while synchronizing with the schedulers. There are several synchronization channels from different schedulers, such as "PulseCountperIns" from Gyro Data Acquisition Scheduler TA, "GyropoweronSignal" and "GyropoweroffSignal" from Gyro Instantiation Scheduler TA, and "GyropowerstateperIns" from Gyro Control Output Scheduler TA. Like a typical sensor, the gyro has two operational states, on and off, corresponding to two locations. There are two self-transitions that update "PulsecountStoIns" to monitor the satellite and "Gyropowerstate" to measure its own state driven by different synchronizations. Ultimately, we obtain the *Gyro* TA in Fig. 2(a). The *sun sensor* is similar to the *gyro* as well as data management computer shown in Fig. 2(b) and (c).

The actuator *triaxial control thruster* is utilized to govern the *Satellite* according to the schedulers' instructions. As indicated in Table **??**, it synchronizes with the *Satellite* through sync channels such as "RDSMIns", "PASMIns", "RASMIns", and "CSMIns", and with the scheduler through syncs like "CSMControl" and "ThrusterpoweronSignal", "ThrusterpoweroffSignal", and "ThrusterpowerstateperIns". The *Triaxial Control Thruster* features on and off locations. Several other locations are added for communication with the *Satellite* and scheduler, and there is a self-transition for monitored data updates. Ultimately, we obtain the Triaxial TA in Fig. 2(d).

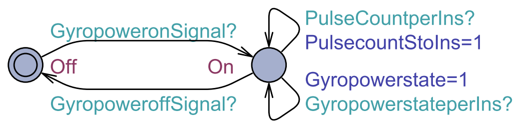
B. Environment modeling

The Environment model used in the Sun Search System comprises three distinct entities: *Satellite*, *Sun*, and *Ground Operator*. These entities belong to different categories. The *Satellite* is a causal domain controlled object, which maintains its orientation towards the *sun* by adjusting its attitude through thruster jets. For the sake of convenience during simulation, we represent changes in satellite attitude by altering its work mode. The *Satellite* possesses four work modes, namely, speed damping (RDSM), tilting search (PASM), rolling search (RASM), and cruise to the sun (CSM), corresponding to four respective locations: RDSM, PASM, RASM, and CSM. The satellite's initial location is RDSM, and it switches between modes through communication channels such as "PASMIns,"

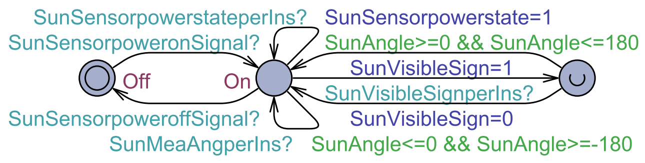

Sun Sensor[i] (SS)
C
Sun(SU)
C
Sun Tracking
Gyro[i](GY)
C
Satellite (SA)
C
Sun Search Controller(SSC)
Triaxial Control Thruster[i](TCT)
C
Data Management Computer(DMC)
C
Ground Operator (GO)
B
Respond to Operator Instruction
a
b
c
d
e
f
g
h
i
j
k
l

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]}

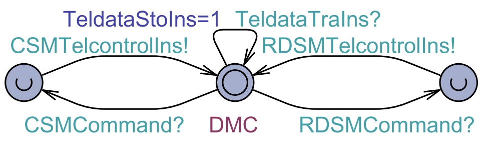
Fig. 1. Sun search system before decomposition - problem diagram



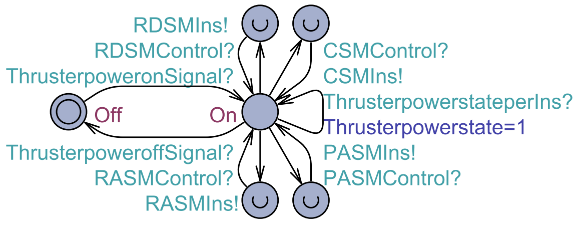
(a) Gyro TA



(b) Sun Sensor TA



(c) Data Management Computer TA



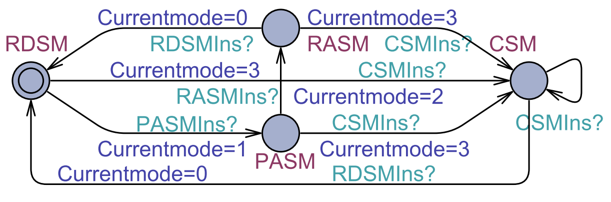
(d) Triaxial Control Thruster TA

Fig. 2. Sun search system device behavior models

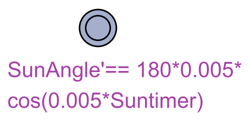
"RDSMIns," "RASMIns," and "Controllingmoment," which are also utilized to communicate with the *Triaxial Control Thruster*. Furthermore, the satellite's rotation is measured through a gyroscope, and it shares the data "AngVelAna". Ultimately, we obtain a *Satellite* TA, shown in Fig. 3 (a).

Regarding the *Sun*, in theory, we ought to model its rotations and the angles that can be observed on the satellite. However, in this simulation, we solely focus on whether or not the Sun is visible. Thus, we establish two locations that correspond to the Sun being visible or invisible, respectively, and transition between the two locations through time constraints. The Sun communicates with the sun sensor via the variable "IsVisible". The Sun TA is ultimately obtained, depicted in Fig. 3(b).

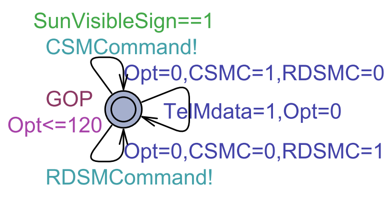
Regarding the Ground Operator, which is a controllable domain, it dispatches instructions to the data management computer through sync channels. Specifically, when the data management computer receives either "CSMCommand" or "RDSMCommand," it indicates that the Ground Operator has dispatched these commands. The valid sequence of these commands yields the corresponding transitions. Moreover, the Ground Operator shares data with the data management computer in the form of "TeleMdata," which is used to designate received data from the data management computer. The Ground Operator TA is depicted in Fig. 3(c).



(a) Satelite TA



(b) Sun TA



(c) Ground Operator TA

Fig. 3. Sun search system Environment model

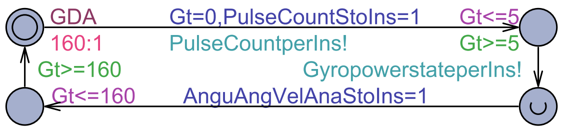
C. Gyro Data Acquisition Scenario

Fig. 4(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. 4(b) shows the Gyro Data Acquisition TA. This TA communicates with *GY* through the channels "PulsecountacqIns" and "GyropowerstateperIns" and with the lexical domain *GD* through the shared data. And in the location Acq1 has a time delay.

```
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. 4. Gyro Data Acquisition

D. Sun Sensor Data Acquisition Scenario

Fig. 5(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. 5(b) shows the Sun Sensor Data Acquisition TA. This TA communicates with *SS* through the channels "SunvisiblesignacqIns", "AngAnaAcqIns", and "SunsensorpowerstateperIns!", and with the lexical domain *SSD* through the shared data. And in the location Acq1 and Acq2 have a time delay.

E. Thruster Data Acquisition Scenario

Fig. 6(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* (causal domain), followed by the storage of data to *TD* (lexical domain) after a 5ms delay.

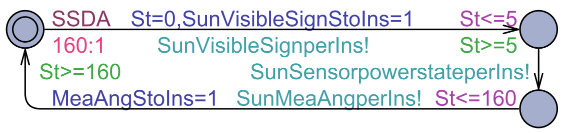
Fig. 6(b) shows the Thruster Data Acquisition TA. This TA communicates with *TCT* through the channel "ThrusterpowerstateperIns", and with the lexical domain *TD* through the shared data. And in the location Acq1 has a time delay.

F. 32ms Interrupt Timer Initialization Scenario

Fig. 7(a) shows cases of a sequence structure that starts the 32ms Interrupt Timer scenario. This scenario involves sending start instruction to *32IT*(causal domain).

```
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

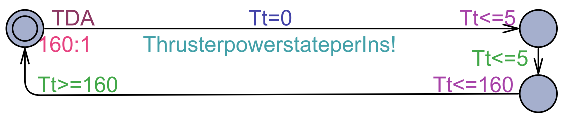


(b) Sun Sensor Data Acquisition TA

Fig. 5. Sun Sensor Data Acquisition

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

(a) Thruster Data Acquisition Scenario



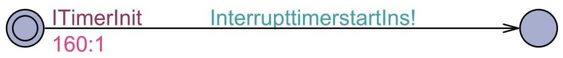
(b) Thruster Data Acquisition TA

Fig. 6. Thruster Data Acquisition

Fig. 7(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.

```
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. 7. 32ms Interrupt Timer Initialization

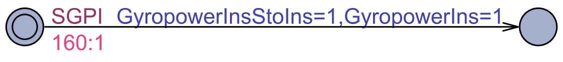
G. Store Gyro Power on Instruction Scenario

Fig. 8(a) shows cases of a sequence structure that saves the Gyro Power on Instruction scenario. This scenario involves sending Gyro Power on Instruction Storage Instruction to *GI*(lexical domain).

Fig. 8(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*.

```
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. 8. Store Gyro Power on Instruction

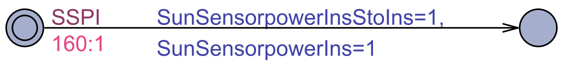
H. Store Sun Sensor Power on Instruction Scenario

Fig. 9(a) shows cases of a sequence structure that saves the Sun Sensor Power on Instruction scenario. This scenario involves sending Sun Sensor Power on Instruction Storage Instruction to *SSI*(lexical domain).

Fig. 9(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*.

```
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. 9. Store Sun Sensor Power on Instruction

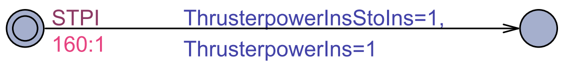
I. Store Thruster Power on Instruction Scenario

Fig. 10(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. 10(b) shows the Store Thruster Power on Instruction TA. This TA communicates with the lexical domain *TPI* 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. 10. Store Thruster Power on Instruction

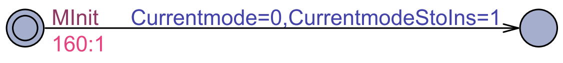
J. Mode Initialization Scenario

Fig. 11(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. 11(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. 11. Mode Initialization

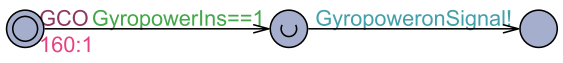
K. Gyro Control Output Scenario

Fig. 12(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. 12(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. 12. Gyro Control Output

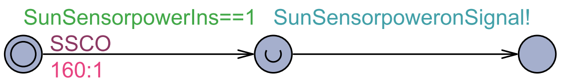
L. Sun Sensor Control Output Scenario

Fig. 13(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. 13(b) shows the Sun Sensor Control Output TA. This TA communicates with *SS* through the channel "Sunsensor-poweronsignal", 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. 13. Sun Sensor Control Output Scenario

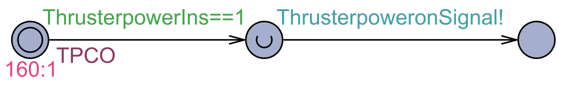
M. Thruster Power Control Output Scenario

Fig. 14(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. 14(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. 14. Thruster Power Control Output

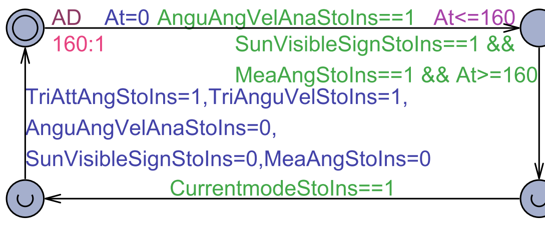
N. Attitude Determination Scenario

Fig. 15(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. 15(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


AD
160:1
At=0
AnguAngVelAnaStoIns==1
At<=160
SunVisibleSignStoIns==1 &&
MeaAngStoIns==1 && At>=160
TriAttAngStoIns=1,TriAnguVelStoIns=1,
AnguAngVelAnaStoIns=0,
SunVisibleSignStoIns=0,MeaAngStoIns=0
CurrentmodeStoIns==1

(b) Attitude Determination TA

Fig. 15. Attitude Determination

O. Mode Switching Management Scenario

Fig. 16(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. 16(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*.

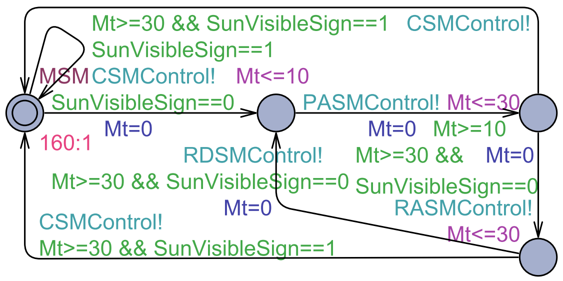
P. Thruster Control Computing Scenario

Fig. 17(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. 17(b) shows the Thruster Control Computing TA. This TA communicates with the lexical domain *ADR* and *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



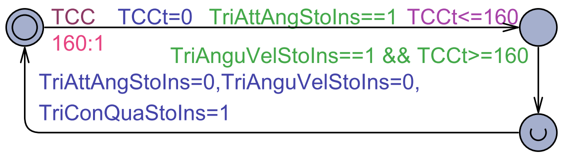
(b) Mode Switching Management TA

Fig. 16. Mode Switching Management

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*.

```
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


TCC
TCCt=0
TriAttAngStoIns==1
TCCt<=160
160:1
TriAnguVelStoIns==1 && TCCt>=160
TriAttAngStoIns=0,TriAnguVelStoIns=0,
TriConQuaStoIns=1

(b) Thruster Control Computing TA

Fig. 17. Thruster Control Computing

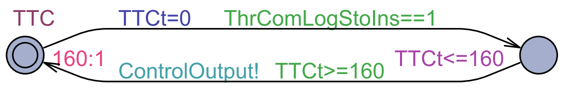
Q. *Thruster Triaxial Control Scenario*

Fig. 18(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).

Fig. 18(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. 18. Thruster Triaxial Control

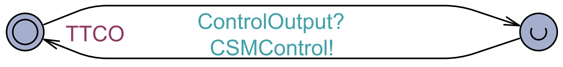
R. *Thruster Triaxial Control Output Scenario*

Fig. 19(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. 19(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 "TriaxialcontrolInsloadIns=1" to denote that the data is loaded from the lexical domain *TTI*.

```
Thruster Triaxial Control Output Scenario{
      TTCO sends TriConInsLoadIns to TTI[event];
      TTCO sends TriConSig to TCT[event];
}
```

(a) Thruster Triaxial Control Output Scenario



(b) Thruster Triaxial Control Output TA

Fig. 19. Thruster Triaxial Control Output

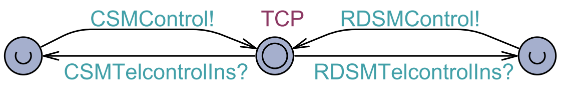
S. Telecontrol Instruction Processing Scenario

Fig. 20(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. 20(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*.

```
Telecontrol Instruction Processing Scenario {
    TIP receives TelIns from DMC[event];
    TIP sends NextCycModeStoIns to MR[event];
}
```

(a) Telecontrol Instruction Processing Scenario



(b) Telecontrol Instruction Processing TA

Fig. 20. Telecontrol Instruction Processing

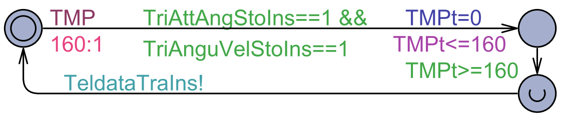
T. Telemetry Processing Scenario

Fig. 21(a) depicts a sequence structure concerning the Telemetry Processing scenario which packages data and sends them to *DMC*(causal domain).

Fig. 21(b) shows the Telemetry Processing TA. This TA communicates with the DMC through the channel "TelDataTransIns", 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*.

```
Telemetry Processing Scenario {
    TP sends CurModeLoadIns to MR[event];
    TP sends TriAttAngLoadIns to ADR[event];
    TP sends TriAngVelLoadIns to ADR[event];
    TP sends TelDataTransIns to DMC[event];
}
```

(a) Telemetry Processing Scenario



(b) Telemetry Processing TA

Fig. 21. Telemetry Processing

U. Simulation and validation

In order to validate our approach, we conducted simulations utilizing a system composed of 3 environment models, 4 system device models, and 19 scheduler models. In the system model declaration within the UPPAAL environment, we specified a total of 26 models. Additionally, we declared 25 synchronization channels, which were derived from all synchronizations within the aforementioned models. Furthermore, we declared 54 global variables, which were sourced from 3 distinct categories. The first category pertains to factual data, such as angles, work modes, and the like. The second category is used to mark the data transit process, for instance, the recording of data being saved into a lexical domain is represented by a variable, where a value of 1 indicates that the save has been completed. The third category comprises clock variables, which are used for time control. Our system contained 15 factual data variables, 33 data transiting mark variables, and 6 clock variables.

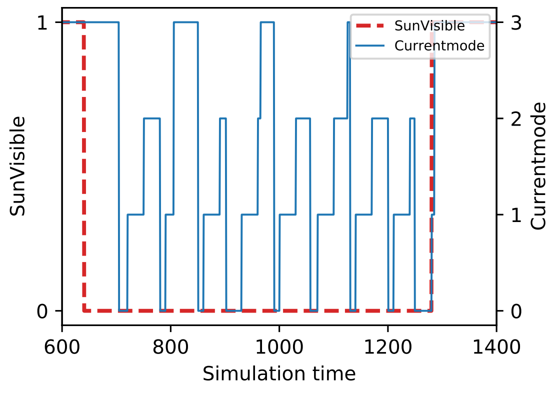
When initializing these variables, we assign factual data values based on real-world observations. For instance, we set the value of the variable IsVisible for the sun to 0, indicating that it is not visible. The current satellite mode model is represented as 0 in UPPAAL, denoting the RDSM mode. Data transition markers are initialized to 0, indicating that there are no data transitions. The clock variables are also initialized to 0 and will be incremented over time.

Once the system models are prepared in terms of NTA, we proceed to simulate them in UPPAAL. In light of our concern regarding the effects on the environment, we execute the following query: simulate[¡=3000]SunVisible, Currentmode, RDSMC, CSMC. The data SunVisible represents the sun's visibility, a value of 0 means the sun is not visible and a value of 1 means the sun is visible. The data Currentmode represents the satellite operating mode, with a value of 0 means in RDSM mode, a value of 1 means in PASM mode, a value of 2 means in RASM mode, and a value of 3 means in CSM mode. The data RDSMC and CSMC represent the commands issued by the ground operator for the satellite to enter RDSM and CSM modes, with a value of 0 indicating that the corresponding command was not issued and a value of 1 indicating that the corresponding command was issued.

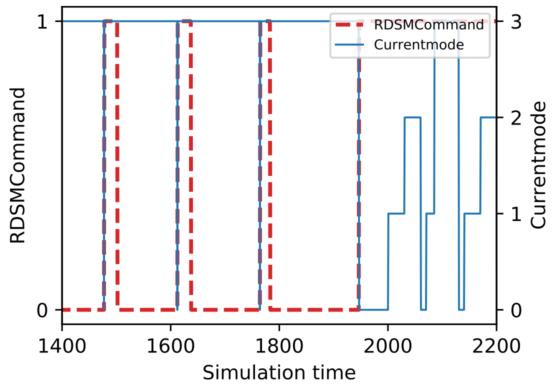
Upon execution, the traces are returned. From these traces, we observe that when the sun is visible, the satellite transitions to CSM mode. Conversely, when the sun is not visible, the satellite operates in either RDSM, PASM, or RASM mode. Additionally, upon receiving a command from the ground operator, the satellite transitions to the corresponding mode. It is therefore evident that the desired effects have been achieved. We can retrieve the necessary values and visualize the effects, as illustrated in Fig. 22.

III. 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 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.



(a) SunVisible - Currentmode



(b) Groud Operator RDSM Command - Currentmode

Fig. 22. Sun Search System Effects Diagram

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

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