An Effect Oriented Simulation Approch for Requirements Validation

***Abstract*—With the pervading of human-cyber-physical scenarios, software is playing a core role in scheduling devices to perform effects on environment for satisfying stakeholders' intentions. However, little effort has been devoted to physical properties of the devices and environment in requirements validation. To address this issue, we propose an effect simulation approach for validating requirements by exploiting the relationships between intentions and device scheduling scenarios. Software is simulated as a device scheduler with a behavior model as well as the devices and environment. The intentions are embodied as effects on environment which is achieved by simulating the interactions between the scheduler, the device models and environment models together. Our approach is demonstrated using a real case from the aerospace domain.**

***Index Terms*—Requirements Validation, Model Simulation, Intention, Device Scheduling Scenarios, Environment Effect**

I. Introduction

With the development and widely use of information technology, human-cyber-physical scenarios are emerging in critical missions that have significant economic and societal importance as well as our daily life [9]. They are commonly seen in smart home, smart cities, intelligent transportation, industrial automation, aerospace etc. In such scenarios, the software system is in the core position [20]. It connects various devices and schedules them to achieve environment effects in order to realize stakeholders' intentions. The software capability is embodied in the device scheduling scenarios [24].

It is essential to validate whether intentions are satisfied through these device scheduling scenarios before they are formally implemented. Intentions often refer to high-level requirements that originate from stakeholders' desires and are typically abstract in nature. On the other hand, device scheduling scenarios are low-level requirements that are concrete and can be implemented. There is a gap between intentions and device scheduling scenarios [7]. It is possible to violate intentions when writing the device scheduling scenarios.

In the validation, the relations between intentions and device scheduling scenarios should be built. The intentions can only be checked by effects on environment. While the scheduling scenarios are describing the software behaviors to schedule the devices. The effects on the environment will be achieved by the devices' effect which must consider the physical properties of both the devices and the environment.

Unfortunately, existing requirements validation approaches does not consider the physical effects and proprieties. In the literature, many requirements validation approaches have been proposed, such as requirements review [17], prototyping [11], user manual development [18], model-based validation [14], and requirements testing [2], none of them explicitly consider the physical properties of the environment and devices. They consider the cyber level, simulating/testing/reasoning the software behaviors [3], [4], [22] or acting as the software prototypes [19], [23]. A most related work is exploring the the relations between scenarios and goals [22] for requirements validation, it simulates the software behaviors to react to the events controlled by participants, rather than environment.

In this paper, we propose an effect driven simulation approach for requirements validation by exploiting the relations between intentions and device scheduling scenarios. Effects are observable phenomena of physical entities that can be detected by humans. They are on devices or environment achieved through interactions between software and devices and between devices and environment. Effects on environment are intentions. They can be predicated by simulating the software behavior model (scheduler for short), the device behavior models, the environment and their interactions together. Then the intention satisfaction could be judged directly by comparing the effects from the intention and the behavior model simulation result.

The contributions of this paper are summarised as follows.

- We propose an model-based functional requirements validation framework by exploiting the relations between intentions and device scheduling scenarios. Our framework is able to simulate behavior models and reason about environment effects.
- We define an effect driven simulation approach for intention validation. It generate scheduler behavior models from device scheduling scenarios, build the device behavior models and the environment behavior models as well as their interactions.

To demonstrate the feasibility of our approach, we conduct a real case study in the spacecraft domain. Specifically, we analyze a sun search subsystem that has 2 intentions and 19 device scheduling scenarios. We simulate the behaviors of 19 schedulers together with 3 devices and 2 environment entities. The intentions are satisfied.

The rest of the paper is organised as follows. Section II compares existing work. Section III introduces description and modeling basis. Section IV defines how to write intentions and device scheduling scenarios. Section IV-C 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. Related Work

The related work involves model-based functional requirements validation. There are many researches using automated reasoning. For example, Aceituna et al. propose a scenario question driven requirements validation [5]. They use state transition diagrams (STD) to model the behavior of software systems, and traverse it using Prolog as reasoning engine to get an answer for the scenario question. They validate whether there are unexpected behaviors which are judged by the users. That approach is applied to an end-of-line product test. Yi et al. present a formal approach to representing and reasoning with goals [3]. It describe goals in terms of actions and static temporal constraints. Business process is described by a set of predicates and formulae describing constraints on the action occurrences. Finally, they rely on logical reasoning to find how to satisfy goals, which is suitable for Information systems.

There are also researches on simulation. For example, Scandurra et al. transform use cases models into Abstract State Machines(ASMs) executable specifications, and then validate them through simulation and scenario-based simulation of the generated ASMs [21]. In the former form of simulation, the generated ASM resembles the system and reacts to the input events of the use case actors. The latter form of simulation allows automatic model-driven animation: scenario are automatically generated from use case model and then animated through simulation. The approach is validated through case studies such as ATM system, an invoice order system.

Some researches focus on generating test cases for validation. For example, Granda et al. propose to combine the testing process with automated reasoning procedures to generate the test cases from a requirements model based on communication analysis [13]. Further, they validate the conceptual schemes according to stakeholders' requirements [14]. A tool CosTest is implemented for transforming instantiation from a Requirements Model into test case. It validate the correctness and completeness of requirements.

Prototypes are also generated from requirements models for requirements validation. For example, Yang et al. [12] present an approach with a developed tool RM2PT [23] to automated prototype generation from formal requirements models for requirements validation. A requirements model consists of a use case diagram, a conceptual class diagram, use case definitions specified by system sequence diagrams, and the contracts of their system operations. They propose a method with a set of transformation rules to decompose a contract into executable parts and nonexecutable parts. The generated prototypes provide a function panel for use case executions, and check whether the system returns the expected results. TestMEReq [19] can automatically generate mock-up user interface as well as abstract test cases from requirements description. It allows multiple stakeholders to collaboratively validate the same set of requirements.

The animation is another important technique for requirements validation [1]. Uchitel et al. present a web animation tool for requirements validation through exploring goals and scenarios [22]. They synthesis a software behavior model in terms of Labelled Transition Systems (LTSs) from scenarios. The animator uses this model based on Labelled Transition Systems Analyser (LTSA) tool to react to events controlled by the animation participants. The events are generated according to the values of fluents used in specifying goals. When a participant initiates an event through the visualizer, if the event cannot be executed at the animator, it indicates that the scenario is inconsistent with the goal. Gabrysiak et al. present a tool for interactively validating requirements through animation [8], which takes BPMN as input and generates a mock-up user interface prototype.

To summarize, the concerns of above researches are the behaviors of software itself. The intentions are either not expressed, or judged by inputs/outputs, or described as goals. They do not consider the physical properties of environment. In human-cyber-physical scenarios, the intentions would be implemented by software through devices to effect on its environment. It is not possible to build the relations between intentions and the device scheduling device scenarios without considering the environment/physical properties. In this paper, we explicitly models the environment, and introduce effects on environment and devices to build the relations between intentions and device scheduling scenarios. We consider the software behaviors as well as the environment behaviors in the simulation. The system simulation will check whether the expected effects could be met.

III. Preliminaries

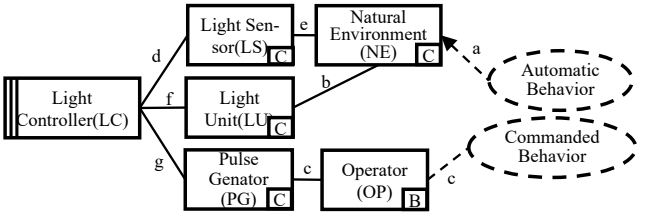
In this section, we use a light controller problem as a running example throughout the paper. *This problem involves an operator, and a light unit which could be turned on and off by receiving OnPulse and OffPulse phenomenon*.

Following Jackson's environment perspective [15], this paper utilizes the basic concepts in Problem Frames (PF) approach [16] as the basic for intention and concept descriptions. Regarding the behavior simulation model, due to the fact that device and environment effects are time-sensitive, we employ timed automata (TA) [10], with UPPAAL [6] serving as the simulation support platform. In the following, we will introduce them.

A. Introduction to basic concepts in PF

In the PF, the environment of the software is assumed to be a set of directly or indirectly interactive problem domains. The problem domains as well as the software are explicitly described in a problem diagram. Fig. 1 is an example of the problem diagram. *Light Controller(LC)* is the $software\ controller$ to be developed, which is represented by a rectangle with double vertical lines. The rectangles connected to the controller are problem domains They are divided into three types: *causal, biddable and lexical domain*. Causal domains (e.g., *Light Unit (LU)*, marked with C) have the predictable causal relationship between their phenomena. A biddable domain (e.g., *Operator (OP)*, marked with B) is usually composed of people, the most important feature is that

it is physical, but there is no clear and predictable internal causality. Lexical domain is the physical representation of data marked with X.



a: NE!{EnvBrightness[value]} b: LU!{Light[state]}
c: OP!{OnButton[event], OffButton[event]} d: LC!{EnvBrightnessAcqIns[event]}
e: LS!{Raw brightness[value]} f: LC!{PowerOnPulse[event], PowerOffPulse[event]}
g: PG!{OnPulse[event], OffPulse[event]}

Fig. 1. Light Controller - problem diagram

The software will interact with *problem domains* which are real-world entities directly (e.g., *Light Sensor (LS)* in Fig. 1) or indirectly (e.g., *Natural Environment(EN)* in Fig. 1). The interaction between the software controller and the problem domain is described by $sender!\{phenomenon[type]\}$. There are three types of phenomenon: event, state and value. An example of an interaction example is $LC!\{onPulse[event]\}$. It means *LC sends onPulse signal to LU*.

A requirement is a description of user wants which is repented by a dotted ellipse *Commanded Behavior* in Fig. 1. Each requirement will constrain or refer expected phenomena on problem domains. These are called requirements references (dotted line) and constraints (dotted arrow) in the PF. There are three types of phenomenon, event, state and value. They mean the expected changes in the problem domains. For example, $LU!\{On[state]\}$, it means *LU* is expected to be *on*.

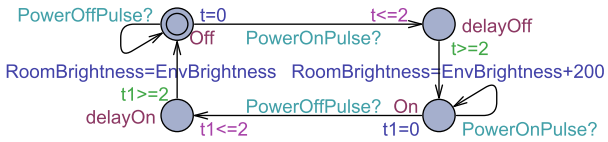
B. Introduction to Behavior model

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 [10].

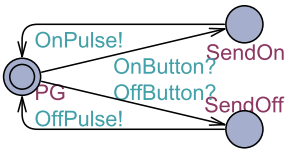
Definition 1 (Timed Automaton): A timed automaton is a tuple $(L, l_0, X, \sum, E, \mathcal{I})$ where: (i) L is a finite set of locations; (ii) $l_0 \in L$ is the initial location; (iii) X is a finite set of clocks; (iv) $\sum$ is a finite set of actions, i.e., synchronisations; (v) $E \subseteq L \times \sum \times \mathcal{G}(X) \times 2^X \times L$ 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<=2* to constrain the value of clock *t*, where *t* can increase over time. The transition from *off* to *delayOff* is triggered by a sync receiving (?) *PowerPulseOn*.

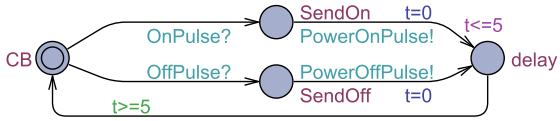
TAs communicate with each other through shared variables and broadcast channels to generate Networks of TAs (NTA) [6]. The shared variables are declared at the system



(a) Light Unit: TA



(b) PulseGenerator



(c) CommandedBehavior: TA

Fig. 2. Two TAs in the Light Controller problem

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 *EnvBrightness* 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.

IV. Notations and our framework

This section firstly describe intentions, effects and device device scenarios. Then, our approach framework is presented.

A. Description of intentions and effects

According to Jackson [15], the requirements are in the environment. They are expected phenomena of the environment. The phenomena on the environment and their orders can describe intentions. For example, in Fig. 1, problem domains *Natural environment (NE)* and *Operator(OP)* are environment. There are two intentions:

1) *automatically adjust the brightness of natural environment to a duration*, and

2) *if OP sends an OnCommand, the brightness of natural environment will be higher, and if OP sends an OffCommand, the brightness of natural environment will be lower.*

The effects are are observable phenomena and their orders on on devices or environment. For example, in the light controller problem, two effects are:

EF1: if OP sends an OnCommand, then the LU is on.

EF2: if OP sends an OffCommand, then the LU is off.

Where *OP sends on OnCommand*, is an effect on *OP*, *LU is On* is a change on *LU*. The former is described as requirements reference in the problem diagram, while the latter is described as an requirement constraint in the problem diagram. In EF1, When the *LU* is on, it will work which adds brightness in

a certain regime. Through this, the brightness will be higer, which means the intention 2) is achieved.

In addition, since the effects are time sensitive, we could also add time constraints on them. For example, EF1 could be: *if OP sends an OnCommand, then within 5s, the LU is on.* Based on this, an intention can also be described using a set of effects on environment. For example, pat of the intention 2) could be: *if OP sends an OnCommand, then within 5s, the brightness is higher.*

B. Description of device scheduling scenarios

Different from common definition of scenarios as sequenced diagram or message sequence chart (MSC), the device scheduling scenarios here mean the scenarios based specification. They describe only software controller behaviors. More specifically, the specification is formed from the projection in the PF [16], [25]. It include a set of scenarios including only software behavior flows.

The scenario has a behavior flow which consists of behavior descriptions and their relations. The behavior is described by basic sentences that the controller sends or receives phenomenon from certain problem domains. It is transformed from the interactions in the problem diagram. For example, in Fig. 3, *the LC receives Time from CL.* There are four behavior relations, sequence (;), loop (while), parallel (||) and choice (if-else if-else) control. In addition, since control is time sensitive, we add two kinds of time constraints. One is at a specific time such as at 5s do something. The other one is the time delay between two behaviors such as after 5ms.

```
Commanded Behavior {
  behavior flows:
    while(true) {
        if(CS receives OnPulse from PG[event]) {
            CS sends PowerOnPulse to LU[event];
        }
        else if(CS receives OffPulse from PG[event]) {
            CS sends PowerOffPulse to LU[event];
        }
        after(5ms);
    }
}
```

Fig. 3. Commanded Behavior scenario

C. Our approach framework

Our approach tries to simulate the scheduler, devices and environment behaviors. We use TA as the behavior model to model them respectively and use shared variables and synchronization to simulate their interactions to form an NTA. Finally, through simulation, effects are obtained to judge whether intentions are satisfied. Therefore, a four-step framework for requirements validation is designed as shown in Fig. 4.

Step 1: generate the scheduler behavior model This step is to generate scheduler behavior models from device scheduling scenarios. Since scenarios are dependent, we argue we can generate a scheduler from each scenario for simulating the behavior of software. As to each scheduler, it will

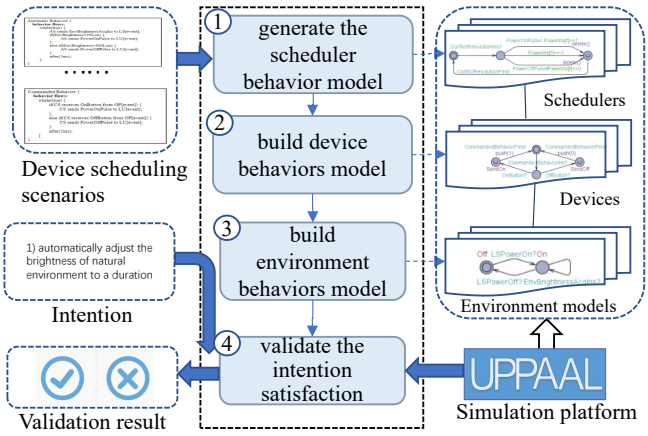


Fig. 4. Framework of our approach

interact with devices directly. We should choose appropriate communication mechanism between them. Behaviors and their sequences specified in the behavior flows will be transformed into locations and transitions in the scheduler TA. This step will result in a set of scheduler TAs.

Step 2: build the device behavior models This step is to build device TAs for all the devices involved in the scenarios. Different device types, e.g., sensors, actuators, should be considered how to construct such TAs. Moreover, the devices not only interact with schedulers, but also interact with environment. So the communication with scheduler TAs and the environment should also be paid attention too. After this step, we will get a set of device TAs.

Step 3: build the environment behavior models This step is to build environment TAs for all the environment involved in the scenarios. Different environment types, e.g., monitored environment entities, controllable entities, should be considered how to construct such TAs. Environment will interact with devices. So we have to consider the communication mechanism with device TAs. After this step, we will get a set of environment TAs.

Step 4: validate the intention satisfaction In this step, we simulate the scheduler TA, device TA, and environment TA together to observe effects on devices and finally on environment. How to compose an NTA that can be executed is the main problem. The intentions described by environment effects are checked upon the simulation results. The problem is how to get After this step, if all the effects described in the intentions are achieved, we claim intentions are satisfied.

V. Our Approach

In this section We explain every step in the framework in detail. During the process, the most important thing is to establish the communication mechanisms between scheduler and devices, devices and environment.

A. Communication mechanism choice

In practice, the schedulers are scheduling devices, Then the devices will be changed and present some effects on them.

In turn, the devices will interact with environment to achieve certain effects. In other words, the effects are by communication of them. According to desired responses and their characteristics, we find different communication mechanisms.

Firstly, we consider the communication between the scheduler and devices. There are two kinds of devices, sensors and actuators. No matter which kinds, and which phenomena they are sharing with the scheduler, we choose the synchronization (one send!, one receive?) mechanism. That is because they should respond to each other quickly, and always wait for the responses to do other things.

For the communication between the devices and environment, situations are complex. In the environment, there are biddable domains (e.g., operator) and causal domains (e.g., natural environment, satellite). The biddable domains only be sensed by sensors. Once they appear, they should be immediately be found and responded. So we still use the synchronization mechanism.

There are two situations that between devices with causal environment. Some causal environment entities are just sharing data with the devices. For example, the brightness between the *light sensor* and the *natural environment*. They only sharing the brightness, so we use shared data as the communication ways. But for some causal environment entities, for example, the satellite, needs to receive signals from the devices. We still use synchronization mechanism.

To summarize, there are different communication mechanisms for different participants as shown in Table I. The interactions between schedulers and devices, between sensors and biddable environment, between devices and causal environment (with event phenomena) are using synchronization mechanism. Only the communication between devices with causal environment (with value or state phenomena) is using shared data.

TABLE I
COMMUNICATION MECHANISMS CHOSEN

Participant I	Participant II	Phenomena type	Communication mechanism
scheduler	device	—	synchronization
scheduler	lexical	—	shared data
sensor	environment (B)	—	synchronization
device	environment (C)	value,state	shared data
device	environment (C)	event	synchronization

B. Scheduler behavior model generation

The scheduler will interact with devices, and behave conforms to the behavior flows in the device scheduling scenarios. All of these will constrain the scheduler TA. We use the synchronization mechanism for communication between devices and schedulers. The communication channel we will follow the behavior flows. So we mainly construct scheduler TA from behavior flows.

In the behavior flows, there are behaviors and their relations, i.e., sequence, choice, loop, and time constraints. They are constraining the scheduler behaviors. Each time a behavior occurs, there can be a location in the scheduler for recording this. So we map each behavior into a location, and their relations into transitions. Different structures have different transitions as shown in Fig. 5:

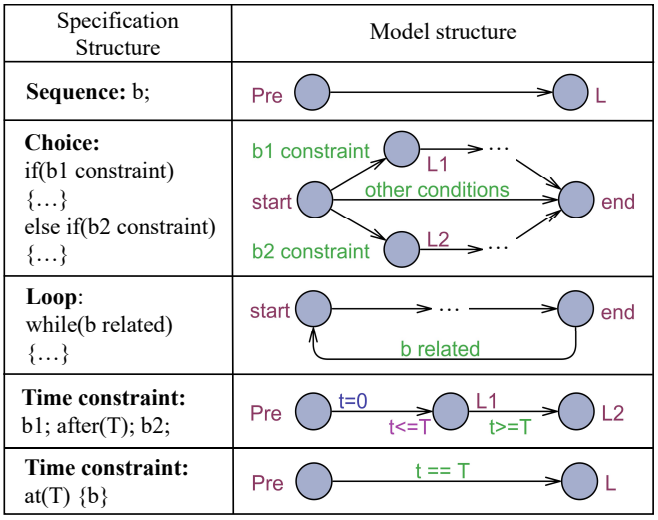


Fig. 5. TA model structures of different scenario structure

- For a sequence like "b;", add a transition directly from the previous location to its corresponding location L;
- For choice like "if (b1 constraint) {$\cdots$} else if (b2 constraint) {$\cdots$})", add 3 paths from the start node to an 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 constraint and b2 constraint will be guard conditions in the transitions.
- For loop like "while (b related) {$\cdots$}", there will be a start and an end node for the {$\cdots$}, 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.

Especially, in the process, the characteristic of behavior b will enforce more information to the transitions as shown in Table II. Usually speaking, the problem domains that interact with the schedulers are devices which are causal domains. In addition to that, there are often designed domains (lexical domains) for saving data. So we only consider two situations:

1) If behavior b is participated by a lexical domain, the communication to a lexical domain is to save or load data, we only need to record whether this operation is done or not. So we add an update like "content(b)=1" in the transition to express that the data operation in b is done.

2) If the behavior b is participated by a device , different types of content of b (content(b)) will indicate different communication mechanisms. If the content is value or state, only the shared variable is able to express it. We adopt the same process with lexical domains. If the content is an event, the communication mechanism would be synced for responding to the event. we use "content(b)Ins!" to express that b is sent by the scheduler, and "content(b)Ins?" to express receiving information.

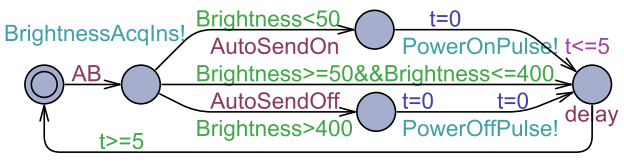
TABLE II
TRANSITION INFORMATION FROM BEHAVIOR CHARACTERISTICS

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 use *Automatic Behavior scenario*as an example to show how to get a scheduler TA. There is a choice structure with two alternative paths (*EnvBrightness<50* and *EnvBrightness>400*) in the scenario (Fig. 6(a)). We find a third path 50<=EnvBrightness<=400 from the initial location to delay location as shown in Fig. 6(b)). For the while structure, there will be a transition from delay location back to the initial location. There is also a time constraint after 5s, we add a guard condition t>=5 in the transition while add an invariant t<=5 at location delay.

```
Automatic Behavior {
  behavior flows:
    while(true) {
        AS sends EnvBrightnessAcqIns to LS[event];
        if(EnvBrightness<50Lux) {
            AS sends PowerOnPulse to LU[event];
        }
        else if(EnvBrightness>400Lux) {
            AS sends PowerOffPulse to LU[event];
        }
        after(5ms);
    }
}
```

(a) Device scheduling scenarios



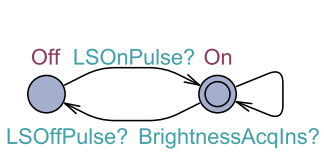
(b) TA

Fig. 6. Light Controller TA: Automatic Behavior

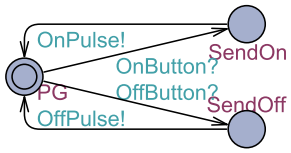
C. System device behavior model construction

In the system device behavior model construction, we also need to consider the communication mechanism both with the scheduler and the environment. In the scheduler, we establish the synchronization way to communicate. The channel should conform to the behaviors in the device scheduling scenarios. For the communication with environment, We have both shared data and synchronization for different situations as shown in Table I. In this section, we consider how to construct device behavior models respect to different kinds of devices, sensors and actuators.

Sensors are used to monitor the attributes of external environment variables periodically or detect humans' behaviors. To build the sensor TA monitoring attributes, we use shared data as communication mechanisms. For example, the *LightSensor* in Fig. 7 shares brightness with natural environment. in addition to the communication mechanisms, we consider the work states and possible work state transitions. We map each work state into a location in TA. Sensors usually have two work states, "off" and "on". They are usually time-driven or event driven. There will be a self-transition updating the environment variables it monitored. When it is acquired by the scheduler, we add an sync message that received from the scheduler. For example, the LightSensor in Fig. 7(a), acquires the environment brightness triggered by receiving (?) a command "EnvBightnessAcqIns".



(a) LightSensor



(b) PulseGenerator

Fig. 7. Light Controller TAs: sensors

For building the sensor TA detecting human behaviors, mapping from scenario to the locations and transitions is similar. The difference is that the communication synchronization is synchronisation for receiving human commands. For example, in Fig. 7(b), the device *PulseGenerator* receives (?) *OnButton* from the *Operator*, and then sends (!) *OnPulse* to the scheduler.

Actuators are usually event driven and change their behaviors by receiving control signals. Their working states are clear such as *on* and *off*. They can be directly mapped to TA locations, and the signals to trigger their changes are mapped to transitions. For example, a *LightUnit* has two working states, *on* and *off*. They correspond to two locations in the *TA* modeling as shown in Fig. 2(a). The state transition from state *off* to *on* in Fig. 2(a) is driven by a channel *"PowerOnPulse"*.

Some of actuators (e.g., *Light Units* Fig. 2(a)) may have impacts on environment attributes, e.g., *environment brightness*. This means the actuators shall interact with the environment through shared data. Furthermore, different states of a device may make different constraints on the natural environment

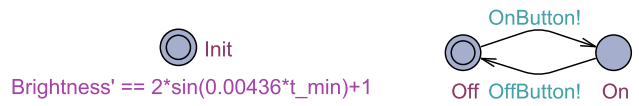
attributes. It is expressed as the invariant of the location such as *EnvBrightness'==1* in Fig. 2(a) at the location "on" while *EnvBrightness'==0* when the *LightUnit* is "off". Usually, there will be response time of actuators. We could add pending locations after they receives signals, but before they turn into expected states. There will be a time invariant such as *t<=2* in Fig. 2(a).

D. Environment behavior model construction

Despite there are three kinds of problem domains in the PF, we only have two kinds of environment, causal and biddable domains. Moreover, the causal domains can be further classified into two kinds: natural environment and controllable objects. The natural environment such as room temperature, can be monitored and changed by the devices. The controllable objects are just like actuators, which could be controlled by receiving signals. The construction of controllable objects are the same with actuators, so we just ignore it.

For building the TA for natural environment, we concern the attributes that it may have. Some attributes, e.g., the room temperature, are continuously changed following a specific law of nature. This will be declared as an invariant in a location as shown in Fig. 8(a). Moreover, we also notice some of the attributes are not always continuous. For example, rain or not rain, we consider them as states, and simulate the process from rain to not rain using time.

For building TA for a biddable domain (mainly humans), we only consider valid commands and their relations. The communication is using synchronization. Each time it sends or receives a valid command, there will be a location. The sequence of the commands will be transitions between them. For instance, the *Operator* in Fig. 8(b), sends (!) *OnButton* and *OffButtton* commands alternatively.



(a) Natural environment (b) Operator

Fig. 8. Light Controller TAs: Environment model

E. Intention validation

The intention validation is checked through effects which are reasoned through simulation. Simulation is supposed to be performed in the simulation platform UPPAAL, In addition to the models, there are two more steps before actual simulations:

Step 1: declare system models: there are many sub-steps:

The model declaration. This step is to declare the scheduler models, the device models, and environment models.

The declaration of broadcast channels. Channels relate to synchronizations, so they can be obtained from scheduler TAs, device TAs, and environment TAs.

The declaration of global variables. We need to declare the shared variables between different TAs, including the identifier of each scheduler TA, the state identifier of device TA, and the environment attributes in environment TA. For the identifiers of scheduler TA, since an identifier is a discrete variables indicating the scheduler triggered or not, we set it as integer variable, such as *"int scheduler1"* for scheduler TA of *scenario1*. For the state identifiers of devices, since a state identifier represents the state of the device, we also set it as integer variable. For the environment attributes, since we consider all the attributes to be continuously changing, we set them as clock variables, such as *clock temperature*. Besides, we also need to declare the change rate variables, e.g., *"dtemper"*. They are *"double"* variables.

The set of initial values Initial values of variables set differently. For the identifier of global variables. For each scheduler TA, we set it to be 0 representing they are not working. For the the state identifier of device TA, we set it to be initial state of the device. For the attributes of environment, we could set them according to the real cases.

Step 2: simulate the system models in UPPAAL

After obtaining the system models in terms of NTA, we simulate them via platform UPPAAL. We execute the following query:

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

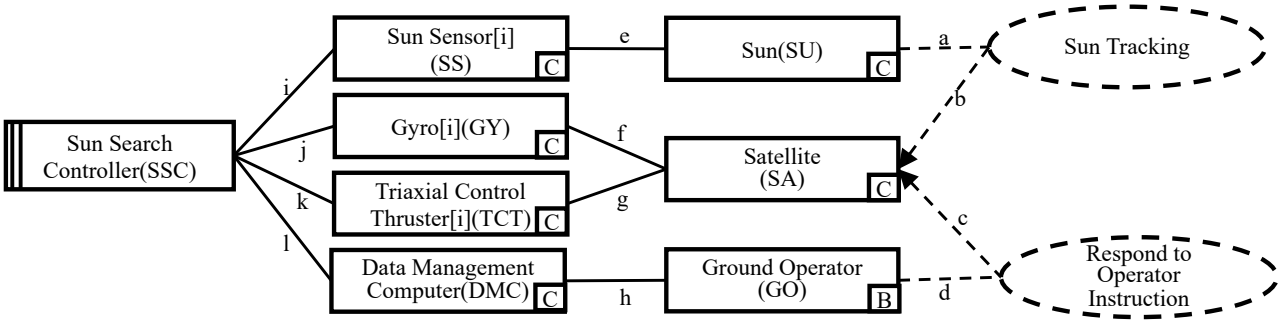
Where t is the simulation time; $a_1, a_2, \cdots, a_i$ are attributes to be monitored, e.g., *temperature* and *light brightness*; and $d_1, d_2, \cdots, d_n$ are the state variables of devices, such as *LightUnit*. The execution result is a set of simulation traces, which record the change over time of the above variables. The simulation trace of each variable takes the form of: *var:* (t_0, v_0),...,(t_i, v_i),...,(t_m, v_m), $i \in [0, m]$, where, *var* is the variable name, t_i is the time point, and v_i is the value of *var*. (t_i, v_i) shows that at the time point t_i, *var* takes the value of v_i. From these variables, we could check whether the expected effects happen in the simulation or not. If they happen, we say the intentions are satisfied.

VI. Real case study

In this paper, 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.*

A. Expected effects and device scheduling scenarios

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. 9 shows its



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!{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. 9. Sun search system before decomposition - problem diagram

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 [16], [25] 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. We choice 3 typical scenarios as shown in Fig. 10. Fig. 10(a) is a sequence structure with time constraint showing the scenario of Gyro Data Acquisition. It collects data from GY (causal domain), and after 5ms saves the data to GD(lexical domain). Fig. 10(b) shows a sequence structure showing thruster Control Computing scenario. It gets input data from ADR (lexical domain) and MR (lexical domain), and saves output data to CCR (lexical domain). Fig. 10(c) shows a choice structure called Thruster Triaxial Control Output scenario. It controls TCT(causal domain) according to data from TTI(lexical domain). Due to limited space, please refer to the technical documents [2] for other scenarios.

```
Gyro Data Acquisition {
  behavior flows:
    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

```
Thruster Control Computing {
  behavior flows:
    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];
}
```

(b) Thruster Control Computing

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

(c) Thruster Triaxial Control Output

Fig. 10. Typical scenarios in the sun search system

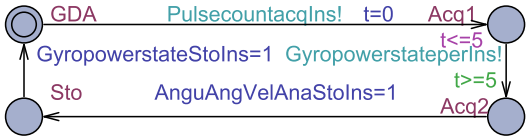
B. Scheduler behavior models

According to the scheduler behavior model generation steps, from the 19 scenarios, we get 19 scheduler behavior models. 18 of 19 scenarios are sequence structure. For the behavior models of the sequence structure, according to model structure in Fig. 5, locations and transitions are direct. To keep them running, they are all back to the initial locations. This is clear in examples in Fig. 11(a) and (b).

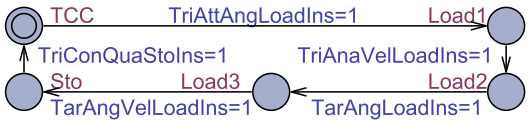
[1]By visible and invisible, it means the sun is at a certain angle. Due to the security needs, we hide them.

[2] https://github.com/jiaozi12/An-Effect-Oriented-Simulation-Approch-for-Requirements-Validation/blob/main/Technical_Documentation.pdf

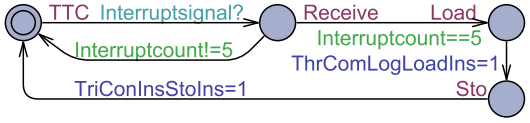
Moreover, despite the sequence structure, since the scheduler are interacting with different types of domains, their communication mechanisms are different. For example, the gyro data acquisition scheduler interact with device gyro and lexical domain GD, according to Table II, the gyro data acquisition scheduler uses synchronization (channel PulseCountAccqIns, GyroPowerStatePerIns) and shared data (AngVelAnaStoIns, GyroPowerStateStoIns). In the thruster control computing scheduler, it only interacts with lexical domains, so it only has one commnication mechanism, i.e., shared data.



(a) Gyro Data Acquisition scheduler TA



(b) Thruster Control Computing scheduler TA



(c) Thruster Triaxial Control scheduler TA

Fig. 11. Scheduler models of sun search system

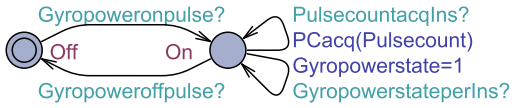
For the behavior models of choice structure, e.g. Fig. 11(b) *ThrusterTriaxialControl*. According to the scenario in Fig. 10(c), the behavior model has only one if branch, and according to the treatment of the branch structure in Fig. 5, we need to generate branches for other conditions. We can see the transition from *Receive* to *Load* correspond to the scenario and the transition from *Receive* to *TTC* is that we added for the simulation model to run correctly. And then process other behaviors based on the transition information shown in Table II.

For the time constraint, we can see the *GyroDataAcquisition* in Fig. 11(c). The scenario Gyro Data Acquisition in Fig. 10(a) has a time constraint after it sends "PulseCountAcqIns" to Gyro. Based on the model structure of the time constraint shown in Fig. 5, we construct this behavior model. It can be seen that there is an invariant "t<=5" at the *Acq1* location, an update condition "t=0" in the transition into the location, and a guard condition "t>=5" in the transition out of the location. Similarly, other behaviors are processed according to Table II.

C. System device behavior modeling

For the system device behavior model, we divided it into sensors and actuators for modeling. The system device behavior model in the sun search system includes the gyro, the sun sensor, the data management computer, and the triaxial control thruster.

The gyro is a sensor and communicates with the scheduler and a causal domain Satellite. According to Table I, it communicates with Satellite through shared data and with the scheduler through synchronization. And the gyro has two work states, on and off, corresponding to two locations. There will be two self-transitions updating the "Pulsecount" and "Gyropowerstate", which are the environment variables it monitored. The sun sensor is a sensor and communicates with the scheduler and a causal domain Sun, the modeling process is similar to the gyro. The data management computer is a sensor and communicates with the scheduler and a biddable domain Ground Operator. According to Table I, it communicates with Ground Operator and the scheduler through synchronization. The data management computer has to receive the sync "TeldataTraIns" from the scheduler and then send the sync "Command" to the ground operator to complete the telemetry processing. Therefore, the location TeleM is added to receive and send these synchronizations. The triaxial control thruster is an actuator and communicates with the scheduler and a casual domain Satellite. According to Table I, it communicates with Satellite and the scheduler through synchronization "TriConSignal" and "Controllingmoment". Similarly, it has on and off locations. Adding Output location for communication with Satellite and scheduler. There is also self-transition for monitored data update.



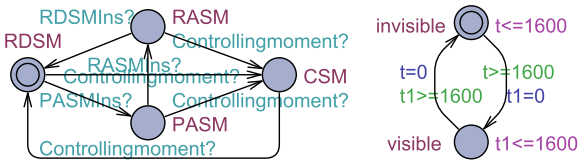
(a) Gyro TA

(b) Data Management Computer TA

Fig. 12. Sun search system Environment models

D. Environment modeling results

The Environment model built in the sun search system contains Sun, Satellite, and Ground Operator. For the Environment model, we can see *Sun* and *Satellite* in Fig. 13. For the *Sun*, we only focus on whether the Sun is visible or not, so we establish two locations corresponding to the Sun being visible and invisible, and transit between the two locations by time constraints. The *Satellite* maintains its orientation to the sun by changing its attitude through the thruster jets. For the convenience of simulation, we represent the change of satellite attitude by changing the work mode of the *Satellite*. The *Satellite* has four work modes: speed damping(RDSM), tilting search(PASM), rolling search(RASM), and cruise to the sun(CSM), corresponding to four locations *RDSM*, *PASM*, *RASM* and *CSM*. And the satellite's initial location is *RDSM* and it switches the work modes through the communication channels "PASMIns", "RDSMIns", and so on.



(a) Satelite (b) Sun

Fig. 13. Sun search system Environment model

E. Simulation and validation

The simulation system includes 3 environment models, 4 system device models, and 19 scenarios. To be able to run the simulation system model correctly, we have to maintain the consistency of the communication channels between these TAs. For example, in Fig. 11(c), the *ThrusterTriaxialControl* sends "TriConSignal" and in Fig. 12(b), the TriaxialControlThruster receives the signal "TriConSignal". And we have to set the initial values of some variables. There are some variables used to mark loading data from the lexical domain or sending data to the lexical domain, for which we need to set the initial value to 0. For example, the *Thruster* needs to load "TriAttAng" from "ADR", we set the variable "TriAttAngLoadIns = 1".

VII. Conclusion

In this paper, we propose to utilize relations between intentions and device scheduling scenarios for requirements validation by effect simulation. We describe the intentions with effects on environment, and achieving such effects by simulating the interactions between schedulers and devices, and between devices and environment. The simulation based validation makes full consideration of the physical properties of devices and environment, which is different from other approaches. The case study on spacecraft domain demonstrates the feasibility of our approach.

In this paper, we also notice that we only consider the environment to be in certain situations. In fact, environment may be full of uncertainty. We will consider that in the future. More case studies will be carried out. Moreover, we plan to develop tool support for the requirements validation.

References

[1] Gemino A. Empirical comparisons of animation and narration in requirements validation. *Requirements Engineering*, 9:153–168, 2004.

[2] Mjeda A and Hinchey M. Requirement-centric reactive testing for safety-related automotive software. In *Requirements Engineering and Testing*, pages 5–8, 2015.

[3] Yi C and Johannesson P. Beyond goal representation: Checking goal-satisfaction by temporal reasoning with business processes. In *International Conference on Advanced Information Systems Engineering*, pages 462–466, 1999.

[4] Aceituna D, Do H, Walia S, and et al. Evaluating the use of model-based requirements verification method: A feasibility study. In *EmpiRE*, pages 13–20, 2011.

[5] Aceituna D, Do H, and Lee W. Sqˆ(2) e: An approach to requirements validation with scenario question. In *Asia Pacific Software Engineering Conference*, pages 33–42, 2010.

[6] Alexandre D, Kim L, Axel L, and et al. Uppaal SMC tutorial. *Int. J. Softw. Tools Technol. Transf.*, pages 397–415, 2015.

[7] Mathias Funk, Lin-Lin Chen, Shao-Wen Yang, and Yen-Kuang Chen. Addressing the need to capture scenarios, intentions and preferences: Interactive intentional programming in the smart home. *International Journal of Design*, 12(1):53–66, 2018.

[8] Gabrysiak G, Giese H, and Seibel A. Deriving behavior of multi-user processes from interactive requirements validation. *Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering*, 2010.

[9] Linkov I and Kott A. Fundamental concepts of cyber resilience: Introduction and overview. *CoRR*, pages 1–25, 2019.

[10] Bengtsson J and Yi W. Timed automata: Semantics, algorithms and tools. In *Advanced Course on Petri Nets*, pages 87–124, 2003.

[11] Kolthoff K. Automatic generation of graphical user interface prototypes from unrestricted natural language requirements. In *34th IEEE/ACM International Conference on Automated Software Engineering*, pages 1234–1237, 2019.

[12] Yang L, Li X, Ke W, and Liu Z. Automated prototype generation from formal requirements model. *IEEE Trans. Reliab.*, 69(2):632–656, 2020.

[13] Granda M, Condori-Fernández N, Vos T, and et al. Towards the automated generation of abstract test cases from requirements models. In *IEEE 1st International Workshop on Requirements Engineering and Testing*, pages 39–46, 2014.

[14] Granda M, Condori-Fernández N, Vos T, and et al. Costest: A tool for validation of requirements at model level. In *IEEE 25th International Requirements Engineering Conference*, pages 464–467. IEEE, 2017.

[15] Jackson M. The meaning of requirements. *Ann. Softw. Eng.*, 3:5–21, 1997.

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

[17] Singh M. Automated validation of requirement reviews: A machine learning approach. In *IEEE 26th International Requirements Engineering Conference*, pages 460–465, 2018.

[18] Garcia N, Lüdtke M, Kortik S, and et al. Bootstrapping MDE development from ROS manual code - part 1: Metamodeling. In *Third IEEE International Conference on Robotic Computing*, pages 329–336, 2019.

[19] Moketar N, Kamalrudin M, Sidek S, and et al. An automated collaborative requirements engineering tool for better validation of requirements. In *31st IEEE/ACM International Conference on Automated Software Engineering*, pages 864–869. ACM, 2016.

[20] Hu P, Ning H, Chen L, and et al. An open internet of things system architecture based on software-defined device. *IEEE Internet of Things Journal*, pages 2583–2592, 2018.

[21] Scandurra P, Arnoldi A, Yue T, and Dolci M. Functional requirements validation by transforming use case models into abstract state machines. In *ACM Symposium on Applied Computing*, 2012.

[22] Uchitel S, Chatley R, Kramer J, and et al. Fluent-based animation: exploiting the relation between goals and scenarios for requirements validation. In *RE*, pages 208–217, 2004.

[23] Yang Y, Ke W, and Li X. Rm2pt: Requirements validation through automatic prototyping. In *IEEE 27th International Requirements Engineering Conference*, pages 484–485. IEEE, 2019.

[24] Jin Z. *Environment Modeling-Based Requirements Engineering for Software Intensive Systems*. Morgan Kaufmann, 2018.

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