Investigating the effects of designing industrial control software using push and poll strategies

Citation for published version (APA):

Document status and date:
Published: 01/01/2011

Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.
Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain.
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 03. Sep. 2020
Investigating the Effects of Designing Industrial Control Software using Push and Poll Strategies

J.F. Groote\textsuperscript{1}, A.A.H. Osaiweran\textsuperscript{1}, M.T.W. Schuts\textsuperscript{2}, and J.H. Wesselius\textsuperscript{2}
\textsuperscript{1} Eindhoven University of Technology, Eindhoven, The Netherlands
\textsuperscript{2} Philips Healthcare, BU Interventional X-ray, Best, The Netherlands
{j.f.groote, a.a.h.osaiweran}@tue.nl, {matijs.schuts, jacco.wesselius}@philips.com

Abstract

In this paper we apply a number of design guidelines for circumventing the state space explosion problem from [J.F. Groote, T.W.D.M. Kouters, and A.A.H. Osaiweran, Specification guidelines to avoid the state space explosion problem, 2011] to the design and formal verification of a real industrial case, namely a controller of a power distribution unit of X-ray machines developed at Philips Healthcare. Through this work we investigate whether these guidelines are effective in designing practical applications. We provide a number of alternative designs that mainly incorporate pushing and polling strategies, taking into account a number of these guidelines. Using the pushing strategy components notify one another when information becomes available while using polling components ask for information only when it is needed. We find that designs that use a pushing strategy and do not apply such guidelines typically lead to the generation of substantially more states. All demonstrated designs formally refine a single predefined external specification that captures the desired external behavior of the system. Moreover, all designs are deadlock free and do not exhibit any illegal interactions. This confirms our hypothesis that the design guidelines are really effective in practical contexts.

1 Introduction

Due to the increasing complexity of industrial control software, establishing the behavioral correctness is a challenging task. During the construction of large industrial software systems, errors are regarded as inevitable and some are often hard to analyze or even to reproduce, due to the concurrent nature of interacting components. Hence, techniques for automatic detection of flaws are widely encouraged, to assist developers building their software rapidly and correctly.

Behavioral verification tools, mainly using model checking technology, can be used for verifying the discrete behavior of complex industrial software designs [13, 3, 4]. They assist correctness verification of designs of complex systems prior to their actual implementation. In a number of reported industrial cases [12, 16], design errors have been discovered, which were hard to find using conventional testing, due to the concurrent nature of the components. Since model checking tools provide high-level automation compared to other verification techniques such as theorem proving, they quickly become more popular and attractive in industry.

Model checking tools require a model or specification that precisely describes the behavior of concurrent components to be verified. The model can thoroughly be investigated to prove that the components always satisfy certain requirements. The tools perform enumerative, systematic exploration of all (or part of) possible execution scenarios of the molded system. The set of the execution scenarios are often characterized by an LTS (label transition system or state space) which contains states and transitions labeled by actions performed by the components.

But the behavioral verification is limited by the state space explosion problem, which arises when the verified components include a huge number of states that cannot fit into memory, de-
spite the use of clever verification algorithms and powerful computers. Although model checking
technologies nowadays available can potentially handle billions of states, they still suffer from this
problem. For some practical cases developers have to wait hours or days for outcomes resulting
from the tools when verifying even a single property of their systems.

In [7, 8] we have proposed a number of guidelines to tackle the state space explosion problem
but in a different manner, namely by designing software components such that they can be easily
verified. In this paper we apply a number of these guidelines on the design and the formal
verification of a practical industrial case, namely a controller of a power distribution unit (PDU)
[10, 11, 9], used for controlling the electrical power, of X-ray machines developed at Philips Health-
care. Through this we want to know whether the guidelines are effective in practical context. To
accomplish this, we propose a number of alternative designs to achieve the required functionality
of the controller. We found that the designs that do not use the guidelines have substantially more
states.

We start by describing a single desired external behavior of the controller. Then, we provide
two main designs, where the first uses a pushing strategy and the second uses a polling strategy.
By pushing we mean that components of a system share their information with others when
the information is available, while polling means that components poll (or ask) information from
others only when it is needed. As will be demonstrated shortly, other guidelines such as the
restrict use of data and the use of global synchronous communication have been further applied
and substantially helped reducing the state space. All design alternatives refine the external
behavior of the controller and provide the intended behavior of the system.

Throughout this article we use mCRL2 [13, 6] for formal specification and state space gener-
ation. Hence, we assume a basic knowledge of the description language and the tool set. Addi-
tionally, we use the refinement concept to prove formal refinement of designs against the external
behavior. For this we use mCRL2, CADP [3] and CSP/FDR2 [15, 4].

The results of this work confirms that different design styles can reduce the number of the gen-
erated states of the modeled systems and that the guidelines are effective in practical applications.

This paper is organized as follows. In Section 2 we bring a list of guidelines used for designing
and verifying the PDU controller from [7, 8]. Section 3 gives an overview of the context of the
PDU controller. The strategies and tactics used to accomplish the tasks of modeling and verifying
the controller are described in Section 4. The external behavior of the controller is detailed in
Section 5. The designs of the controller using the pushing strategy are demonstrated in Section 6,
while the designs implementing the poll strategy are described in Section 7. In Section 8 we give
some statistical data, comparing the push and poll variants and the used tools.

2 Overview of the used guidelines

In this section we give a concise description of the guidelines [7, 8] that we used in this paper.

1. Information polling. This guideline advises to let processes ask for information, whenever
it is required. The alternative is to share information with other components, whenever the
information becomes available. Although, this latter strategy clearly increases the number
of states of a system, it appears to prevail over information polling in most specifications
that we have seen.

2. Global synchronous communication. If more parties communicate with each other, it
can be that a component 1 communicates with a component 2, and subsequently, component
2 informs a component 3. This requires two consecutive communications and therefore two
state transitions. By using multi-actions it is possible to let component 1 communicate with
component 2 that synchronously communicates with a component 3. This only requires one
transition. By synchronizing communication over different components, the number of states
of the overall system can substantially be reduced.

3. Avoid parallelism among components. If components operate in parallel, the state
space grows exponentially in the number of components. By sequentializing the behavior
of these components, the size of the total state space is only the sum of the sizes of the state spaces of the individual components. In this latter case state spaces are small and easy to analyze, whereas in the former case analysis might be quite hard. Sequentializing the behavior can for instance be done by introducing an arbiter, or by letting a process higher up in the process hierarchy to allow only one sub-process to operate at any time.

4. **Restrict the use of data.** The use of data in a specification is a main cause for state-space explosion. Therefore, it is advisable to avoid using data whenever possible. If data is essential, try to categorize it, and only store the categories. For example, instead of storing a height in millimeters, store too_low, right_height and too_high. Finally, take care that data is only stored in one way. E.g., storing the names of the files that are open in an unordered buffer is a waste. The buffer can be ordered without losing information.

5. **Specify the external behavior of sets of sub-components.** If the behavior of sets of components are composed, the external behavior tends to be overly complex. In particular the state space is often larger than needed. A technique to keep this behavior small is to separately specify the expected external behavior first. Subsequently, the behaviors of the components are designed such that they meet this external behavior.

The following two guidelines are not used in this work, but it is worth mentioning them here for the sake of completeness.

1. **Confluence and determinacy.** When parallel behavior cannot be avoided, it is useful to model such that the behavior is \( \tau \)-confluent. In this case \( \tau \)-prioritisation can be applied when generating the state space, substantially reducing the size of the state space. Modeling a system such that it is \( \tau \)-confluent is not easy. A good strategy is to strive for determinacy of behavior. This means that the ‘output’ behavior of a system must completely be determined by the ‘input’. This is guaranteed whenever an internal action (e.g. receiving or sending a message from/to another component) can be done in a state of a single component, then no other action can be done in that state.

2. **Compositional design and reduction.** If a system is composed out of more components, it can be fruitful to combine them in a stepwise manner, and reduce each set of composed components using an appropriate behavioral equivalence. This works well if the composed components do not have different interfaces that communicate via not yet composed components. So typically, this method does not work when the components communicate in a ring topology, but it works very nicely when the components are organized as a tree.

3 **The context of the PDU controller**

We start by illustrating the context of the PDU controller. Philips healthcare, at Best, the Netherlands is developing a family of highly sophisticated, computerized X-Ray systems. The systems include a distributed architecture in the sense that clinical applications, required for establishing X-Ray examinations, are deployed on a cluster of PCs and devices. These components require an efficient and reliable source of power control.

In order to efficiently control the flow of power and to systematically start-up and shutdown the PCs and the devices in an orderly fashion, the system utilizes a Power Distribution Unit (PDU), see Figure 1. All PCs and devices are attached to the PDU. The clinical user has no means of powering on/off the components of the system separately without using the PDU. The clinical user can only initiate start-up and shutdown requests by pushing a number of buttons on a user console attached to the PDU. Upon pressing these buttons the PDU controls the flow of power to the components.

The PDU console provides two buttons: PowerOn, and PowerOff. The PDU includes an optional EmergencyOff button which can be used to cut down any source of power to the system,
in case of calamities. This optional button may be installed on request or when mandatory by legislation.

To manage the start-up and shutdown behavior of the system, the PDU employs two networks. The first network is the power network which supplies the attached PCs and devices with the necessary power. The flow of power to the components is controlled by the PDU by switching a number of power taps on and off. The second network is an Ethernet network, by which the PDU can communicate with the PCs and the devices, through a number of dedicated signals.

The PDU taps are classified into switchable and permanent taps. The switchable taps can be switched on/off by the PDU. The permanent taps are powered when the system is off in the perception of the clinical user, but not in the perception of the PDU (the system is in standby from perspective of the PDU). This allows the attached components to be available for batch processing, maintenance and remote accesses purposes. The permanent taps can be switched off when forced by the clinical users (e.g., by pressing the EmergencyOff button).

**PCs and devices** The PCs and devices depicted in Figure 1 almost expose the same start-up and shutdown behavior, but there is a small difference between the GeoPC (Geometrical PC) and the ControlPC from others, see the state machines in Figure 2.

Initially, a PC is in the Off state. When it is supplied with power, it transits to the StartingUp state where the Operating System (OS) boots up and then the clinical applications are started. After the OS and the applications are up-and-running, the PC transits to the Operational state.

The applications of a PC are restarted upon receiving a restart message from the PDU in Operational state. Additionally, when a shutdown message is received from the PDU, the PC
stops all running applications and shuts down the OS.

The GeoPC and the ControlPC include additional behavior. The main function of the GeoPC is controlling a number of motorized movable segments such as the table where patients lay on and the stands holding X-Ray generators and detectors. On multiple places, the system is equipped with Stop buttons which can be pressed by the users to stop any motorized movement in case of dangerous situations. Upon pressing these buttons the GeoPC sends a CB.stop signal requesting the PDU to switch off the taps to the movable segments. This is visualized in the state machine of GeoPC in Figure 2.

The ControlPC can send a CB.controlPowerOff signal, demanding and forcing the PDU to systematically power off the entire system, including the ControlPC itself. The ControlPC is attached to a permanent tap while all other PCs are connected to switchable taps.

**Behavior of the Power Distribution Unit** The PDU includes a controller that implements the state machine of Figure 3. We assume that the PDU, all PCs and devices are well functioning; therefore, all error scenarios and recovery operations are excluded from the state machine.

![Figure 3: The high-level behaviour of the PDU](image)

The state machine in Figure 3 distinguishes the following six stable states as described in Table 1, and the two transiting states as described in Table 2.

The PowerOn, PowerOff, and EmergencyOff buttons on the user console plus the Stop button of the movable segments, leads to commands that are processed by the PDU controller. Depending on the current state and the supplied command, the controller sends messages to the PCs over the Ethernet network and/or switches the taps on and off if required.

The state machine includes eight distinct events in total. The PDUswitchOn and PDUswitchOff indicate switching the mains disconnector switch on and off, respectively. The powerOff event indicates that the user presses the PowerOff button for less than 3 seconds, while the forcedPowerOff event denotes that the user presses the same button more than 10 seconds. Both powerOn and emergencyOff represent pressing the PowerOn and EmergencyOff buttons, respectively. ControlPowerOff and stop events indicate receiving callback signals from both the ControlPC and the GeoPC, where the first requests the PDU to power off the complete system and the second demands the PDU to immediately cut down the power to the movable segments.
The mains disconnector switch is open which means that the PDU is powerless. All PCs and devices are off.

The permanent power taps are powered. All switchable taps are powerless. ControlPC is on.

All permanent and switchable taps are powered. All PCs and devices are on.

Similar to the System_On state, only the movable parts are powerless. All PCs and devices are on. Motorized movements are disabled.

All permanent and switchable taps are powerless. All PCs and devices are off.

All permanent and switchable taps are powerless. All PCs and devices are off.

<table>
<thead>
<tr>
<th>State</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDU_Off</td>
<td>The mains disconnector switch is open which means that the PDU is powerless. All PCs and devices are off.</td>
</tr>
<tr>
<td>System_Standby</td>
<td>The permanent power taps are powered. All switchable taps are powerless. ControlPC is on.</td>
</tr>
<tr>
<td>System_On</td>
<td>All permanent and switchable taps are powered. All PCs and devices are on.</td>
</tr>
<tr>
<td>GEO_Stop</td>
<td>Similar to the System_On state, only the movable parts are powerless. All PCs and devices are on. Motorized movements are disabled.</td>
</tr>
<tr>
<td>System_Off</td>
<td>All permanent and switchable taps are powerless. All PCs and devices are off.</td>
</tr>
<tr>
<td>Emergency_Off</td>
<td>All permanent and switchable taps are powerless. All PCs and devices are off.</td>
</tr>
</tbody>
</table>

Table 1: The stable states of the PDU state machine [11]

<table>
<thead>
<tr>
<th>State</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartingUpControlPC</td>
<td>The permanent power taps are powered. The ControlPC is starting up.</td>
</tr>
<tr>
<td>StartingUpAllPCs</td>
<td>All permanent and switchable taps are powered. Not all PCs or devices are fully operational.</td>
</tr>
</tbody>
</table>

Table 2: The transitioning states of the PDU state machine [11]

Table 3 summarizes the required tasks for each transition of the state machine. For example, when the system is in the System_Off state and the user presses the PowerOn button, all permanent and switchable taps are switched on, and therefore all PCs and devices start-up. Eventually, all PCs and devices are started-up and the system can potentially move to the System_On state.

In the System_On state, if the user again presses the PowerOn button, the PDU broadcasts a restart message over the Ethernet network. Consequently, the PCs and devices shall restart their applications. But, if the user presses the PowerOff button for less than 3 seconds, the PDU broadcasts a shutdown message over the Ethernet network. Upon receiving the message by the PCs, they gradually shutdown their applications and then their OS. When all PCs and devices are shutdown, the taps will be made powerless by the PDU.

Beside the above mentioned events we introduce a number of indication callback events that reflect the status (or modes) of the system:

- the startingUp event informs external users that the system is in the process of starting up its components,
- the systemStandby event notifies the user that the system is in the System_StandBy state,
- the off event tells the users that the entire system is off,
- the systemOn event informs the user that the system is up-and-running and fully operational,
- and the geoStop event indicates the user that all motorized movements are disabled.

4 Strategy and tactics

The conceptual structure of the specification of the PDU controller is depicted in Figure 4. The external behavior of the PDU and the PCs are depicted as ovals. The design of the PDU controller
<table>
<thead>
<tr>
<th>Transition</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Boot PDU; the PDU switches on all permanent power taps; the ControlPC is starting up.</td>
</tr>
<tr>
<td>2</td>
<td>The PDU switches on all switchable taps, one by one to avoid a big inrush current; all devices are starting up.</td>
</tr>
<tr>
<td>3</td>
<td>The PDU broadcasts a &quot;shutdown&quot; message to shutdown all control devices except the ControlPC; the PDU switches off all switchable taps when power load is below a threshold or when the timer expires.</td>
</tr>
<tr>
<td>4</td>
<td>The PDU immediately switches off all power taps.</td>
</tr>
<tr>
<td>5</td>
<td>The PDU broadcasts a &quot;shutdown&quot; message to shutdown all control devices including the ControlPC; the PDU switches off all taps when power load is below threshold or when the timer expires.</td>
</tr>
<tr>
<td>6</td>
<td>The PDU switches on all taps, one by one to avoid a big inrush current; all devices are starting up.</td>
</tr>
<tr>
<td>7</td>
<td>The PDU broadcasts a &quot;restart&quot; message; the applications of all control devices are restarted.</td>
</tr>
<tr>
<td>8</td>
<td>Disconnect the PDU internal power bus.</td>
</tr>
<tr>
<td>9</td>
<td>The PDU switches on all taps, one by one to avoid a big inrush current; all devices are starting up.</td>
</tr>
<tr>
<td>10</td>
<td>The PDU switches off the power taps that supply motor drives of movable parts.</td>
</tr>
<tr>
<td>11</td>
<td>The PDU switch on the power taps that supply motor drives of movable parts.</td>
</tr>
<tr>
<td>12</td>
<td>The PDU is switched off; all taps are switched off.</td>
</tr>
</tbody>
</table>

Table 3: The activities required for each transition of the PDU state machine [11]

is shown as a square shape. The communication channels with the direction of information flow are depicted using arrows.

The combined model refines the external specification of the PDU. It must be deadlock, livelock and illegal free.

Figure 4: Conceptual structure of the specification of the PDU controller

The figure shows the structure of a combined model that includes the parallel composition of the PDU controller and the external specification of the PCs, highlighting the communication channels used for exchanging information among the components. Each design alternative of the PDU controller has a different combined model. To construct these models we have followed a number of steps, summarized below.
Modeling the external behavior of the PDU  First, we modeled the desired external behavior of the PDU with respect to the external users of the system. This specification includes all external commands issued by the user console plus all indication callback signals sent to the user. The specification is identical for all design alternatives, and is used as a guide for implementing the alternative designs we are comparing. This external behavior excludes any internal interaction with the PCs.

Describing the external behavior of the PCs  The external behavior of each PC is described with respect to the interaction required with the PDU. The description excludes any activities performed internally by the PC.

Constructing alternative designs for the PDU controller  We design the PDU controller in two manners, namely a design where PCs ‘push’ their information to the PDU when the information is available, and another design where the PDU ‘polls’ information from the PCs whenever it is required.

For each design manner there are a number of alternatives that assist further reducing the state space. All design alternatives adhere to the external specification, and provide the external users of the system with the expected behavior.

Modeling conventions  In the specification of all models any action pre-fixed by the letter ‘r’ denotes the receiving party of a communication whereas actions pre-fixed by ‘s’ denote the sending party. The result of a communication is denoted by an action without any pre-fixed letter.

Specification completeness  In every state of the external behavior of the PCs we assign illegal responses to the stimuli if they are not expected in a state. The same response is assigned to callbacks received from the PCs in the specification of the PDU design for detecting unexpected callbacks. During the behavioral verification we search for the occurrences of such an event plus deadlock and livelock scenarios.

Refining the external behavior  Each design alternative is checked against the external specification using a number of refinement models: weak-trace [2], Failures [15], Failures-divergences [15], observational [14], safety [1], Tau* [5] and branching-bisimulation [17]. The reason of choosing refinement over equivalence check is that checking equivalence may tend to be overly complex. It may require that both the implementation and the external behavior to strictly have the same structure, so the external specification might be forced to be adjusted to satisfy the structure of the design. This is what we are trying to avoid here.

Instead of using equivalence checks we prove refinement of designs by means of inclusion (or preorder) checks. Precisely, we prove that the behavior of a design is included in the behavior of the external specification. Upon the success of the check we know that the design always exposes expected behavior to the external world under the refinement model being used, i.e., no extra unexpected behavior would result from the concrete implementation of the design crossing the external boundary.

We believe that specifying the external behavior of a system prior to its implementation assists constructing the system better, but does not guarantee building the internal behavior of the system correctly. Checking correctness of internal behavior of systems can be accomplished by other means such as searching for deadlocks, livelocks, illegal interactions and verifying properties on systems.

The details of the steps performed throughout this case are addressed in the subsequent sections.

5  The external specification of the PDU controller

We started our modeling activities by considering the fifth guideline. The external specification of the PDU controller in the mCRL2 language is listed below. It precisely describes the external
behavior of the PDU, with respect to the external users, reflecting the internal modes of the system using states and visible indication callbacks, matching the state machine of Figure 3. It includes all user commands as input stimuli, and all user indication callbacks as responses to the external world. It excludes all internal interactions such as internal system messages and powering on/off the PCs.

\[
\begin{align*}
&\text{proc ExtSpec(s:State)=} \\
&\quad \begin{cases} 
&\quad (s==\text{PDU}_\text{Off}) \rightarrow \\
&\quad \quad \begin{cases} 
&\quad \quad \text{(IPDU(PDUswitchOn) . IndicationCB(startingUp) . ExtSpec(StartingUpCrPC) ) +} \\
&\quad \quad \text{(IPDU(PDUswitchOff) . ExtSpec(PDU}_\text{Off}) +} \\
&\quad \quad \text{IPDU(powerOn) . IndicationCB(startingUp) . ExtSpec(StartingUpAllPCs) +} \\
&\quad \quad \text{IPDU(powerOff) . ExtSpec(System}_\text{StandBy) +} \\
&\quad \quad \text{IPDU(forcedPowerOff) . IndicationCB(off) . ExtSpec(System}_\text{Off}) +} \\
&\quad \quad \text{IPDU(emergencyOff) . IndicationCB(off) . ExtSpec(Emergency}_\text{Off}) +} \\
&\quad \quad \text{int . IndicationCB(off).ExtSpec(System}_\text{Off} ) + \\
&\quad \end{cases} \\
&\quad \end{cases} \\
&\quad \begin{cases} 
&\quad (s==\text{System}_\text{StandBy}) \rightarrow \\
&\quad \quad \begin{cases} 
&\quad \quad \text{(IPDU(PDUswitchOff) . ExtSpec(PDU}_\text{Off}) +} \\
&\quad \quad \text{IPDU(powerOn) . IndicationCB(startingUp) . ExtSpec(StartingUpAllPCs) +} \\
&\quad \quad \text{IPDU(powerOff) . IndicationCB(System}_\text{StandBy) . ExtSpec(System}_\text{StandBy}) +} \\
&\quad \quad \text{IPDU(forcedPowerOff) . IndicationCB(off) . ExtSpec(System}_\text{Off}) +} \\
&\quad \quad \text{IPDU(emergencyOff) . IndicationCB(off) . ExtSpec(System}_\text{Off} +} \\
&\quad \quad \text{int . IndicationCB(off).ExtSpec(System}_\text{Off}) +} \\
&\quad \quad \begin{cases} 
&\quad \quad \text{int . IndicationCB(System}_\text{On}) . ExtSpec(System}_\text{On}) +} \\
&\quad \quad \text{int . ExtSpec(StartingUpAllPCs) +} \\
&\quad \quad \text{int . IndicationCB(geoStop) . ExtSpec(geoStop) ) +} \\
&\quad \end{cases} \\
&\quad \end{cases} \\
&\quad \begin{cases} 
&\quad (s==\text{System}_\text{On}) \rightarrow \\
&\quad \quad \begin{cases} 
&\quad \quad \text{(IPDU(PDUswitchOff) . ExtSpec(PDU}_\text{Off}) +} \\
&\quad \quad \text{IPDU(powerOn) . IndicationCB(startingUp) . ExtSpec(StartingUpAllPCs) +} \\
&\quad \quad \text{IPDU(powerOff) . IndicationCB(systemStandby) . ExtSpec(System}_\text{StandBy} +} \\
&\quad \quad \text{IPDU(forcedPowerOff) . IndicationCB(off) . ExtSpec(System}_\text{Off}) +} \\
&\quad \quad \text{IPDU(emergencyOff) . IndicationCB(off) . ExtSpec(System}_\text{Off} +} \\
&\quad \quad \text{int . IndicationCB(off).ExtSpec(System}_\text{Off}) +} \\
&\quad \quad \begin{cases} 
&\quad \quad \text{int . IndicationCB(geoStop) . ExtSpec(geoStop) ) +} \\
&\quad \end{cases} \\
&\quad \end{cases} \\
&\quad \begin{cases} 
&\quad (s==\text{StartingUpAllPCs}) \rightarrow \\
&\quad \quad \begin{cases} 
&\quad \quad \text{(IPDU(PDUswitchOff) . ExtSpec(PDU}_\text{Off}) +} \\
&\quad \quad \text{IPDU(powerOn) . IndicationCB(startingUp) . ExtSpec(StartingUpAllPCs) +} \\
&\quad \quad \text{IPDU(powerOff) . IndicationCB(System}_\text{StandBy) . ExtSpec(System}_\text{StandBy}) +} \\
&\quad \quad \text{IPDU(forcedPowerOff) . IndicationCB(off) . ExtSpec(System}_\text{Off}) +} \\
&\quad \quad \text{IPDU(emergencyOff) . IndicationCB(off) . ExtSpec(System}_\text{Off} +} \\
&\quad \quad \text{int . IndicationCB(geoStop) . ExtSpec(geoStop) ) +} \\
&\quad \end{cases} \\
&\quad \end{cases} \\
&\quad \begin{cases} 
&\quad (s==\text{Geo}_\text{Stop}) \rightarrow \\
&\quad \quad \begin{cases} 
&\quad \quad \text{(IPDU(PDUswitchOff) . ExtSpec(PDU}_\text{Off}) +} \\
&\quad \quad \text{IPDU(powerOn) . IndicationCB(systemOn) . ExtSpec(System}_\text{On}) +} \\
&\quad \quad \text{IPDU(powerOff) . IndicationCB(System}_\text{StandBy) . ExtSpec(System}_\text{StandBy}) +} \\
&\quad \quad \text{IPDU(forcedPowerOff) . IndicationCB(off) . ExtSpec(System}_\text{Off}) +} \\
&\quad \quad \text{IPDU(emergencyOff) . IndicationCB(off) . ExtSpec(System}_\text{Off} +} \\
&\quad \quad \text{int . IndicationCB(geoStop) . ExtSpec(geoStop) ) +} \\
&\quad \end{cases} \\
&\quad \end{cases} \\
&\quad \begin{cases} 
&\quad (s==\text{System}_\text{Off}) \rightarrow \\
&\quad \quad \begin{cases} 
&\quad \quad \text{(IPDU(PDUswitchOff) . ExtSpec(PDU}_\text{Off}) +} \\
&\quad \quad \text{IPDU(powerOn) . IndicationCB(startingUp) . ExtSpec(StartingUpAllPCs) +} \\
&\quad \quad \text{IPDU(powerOff) . ExtSpec(System}_\text{Off}) +} \\
&\quad \quad \text{IPDU(forcedPowerOff) . ExtSpec(System}_\text{Off}) +} \\
&\quad \quad \text{IPDU(emergencyOff) . ExtSpec(System}_\text{Off} +} \\
&\quad \end{cases} \\
&\quad \end{cases} \\
&\quad \begin{cases} 
&\quad (s==\text{Emergency}_\text{Off}) \rightarrow \\
&\quad \quad \begin{cases} 
&\quad \quad \text{(IPDU(PDUswitchOff) . ExtSpec(PDU}_\text{Off}) +} \\
&\quad \quad \text{IPDU(powerOn) . IndicationCB(startingUp) . ExtSpec(StartingUpAllPCs) +} \\
&\quad \quad \text{IPDU(powerOff) . ExtSpec(Emergency}_\text{Off}) +} \\
&\quad \quad \text{IPDU(forcedPowerOff) . ExtSpec(Emergency}_\text{Off}) +} \\
&\quad \quad \text{IPDU(emergencyOff) . ExtSpec(Emergency}_\text{Off} +} \\
&\quad \end{cases} \\
&\quad \end{cases} \\
&\quad \begin{cases} 
&\quad (s==\text{StartingUpCrPC}) \rightarrow \\
&\quad \quad \begin{cases} 
&\quad \quad \text{(IPDU(PDUswitchOff) . ExtSpec(PDU}_\text{Off}) +} \\
&\quad \quad \text{IPDU(powerOn) . ExtSpec(StartingUpCrPC) +} \\
&\quad \quad \text{IPDU(powerOff) . ExtSpec(StartingUpCrPC) +} \\
&\quad \quad \text{IPDU(forcedPowerOff) . IndicationCB(off) . ExtSpec(System}_\text{Off}) +} \\
&\quad \quad \text{IPDU(emergencyOff) . IndicationCB(off) . ExtSpec(Emergency}_\text{Off}) +} \\
&\quad \quad \text{int . IndicationCB(System}_\text{StandBy) . ExtSpec(System}_\text{StandBy}) ) \\
&\quad \end{cases} \\
&\quad \end{cases}
\end{align*}
\]

To briefly explain the model we choose the \textit{System}_\textit{On} state as an example. The state includes seven summands in total. It precisely describes that when the PDU is in the \textit{System}_\textit{On} state, it can receive any external command from the users. This is indicated by the first five summands. Upon receiving an external command the PDU may send indication callback signals and then transits to a next state. For example, when the PDU receives the \textit{powerOff} command, it sends the \textit{systemStandby} indication to the external users and then transits to the \textit{System}_\textit{StandBy} state.

The last two summands of the state represent the cases where external users can receive indications that the system is off or transiting to the \textit{Geo}_\textit{Stop} state, due to some internal interactions with the PDU. Both \textit{int} events represent detailed activities performed by the concrete implementation of the PDU. For example, \textit{int.IndicationCB(off)} represents the following internal activities:

1. The user of the ControlPC has requested the PDU to power off the entire system via the \textit{controlPowerOff} callback event.
2. The PDU treats the signal by sending the shutdown message around to all devices.
3. The PDU switches all taps off.
4. The PDU sends the IndicationCB(off) signal to the external world.
5. The PDU transits to the System_Off state.

The same technique had been applied to all states of the PDU, matching the original state machine of Figure 3. The complete specification of the model is listed in A. When the specification of the model was completed, it was checked for absence of deadlocks and livelocks. The corresponding LTS had been generated, and used at later stages for the refinement check against the concrete designs of the PDU using mCRL2 and CADP.

6 Implementing the PDU controller using the push strategy

In this variant, the design of the controller utilizes a pushing strategy, in the sense that all PCs share information with the PDU controller upon changes in their internal states. This is illustrated in the sequence diagram in Figure 5. For instance, when the PDU is in the System_On state and the Stop button is pressed, the GeoPC notifies the PDU controller by sending the stop callback event. The same applies to the controlPowerOff callback from the ControlPC. Furthermore, when the PCs are powered on by the PDU, the PDU waits for callbacks from the PCs indicating that they are ready and fully operational.

![Sequence diagram](image)

Figure 5: Example of a scenario where pushing is used

6.1 The external behavior of the PCs

In this section we introduce the external specification of the ControlPC that describes the external behavior with respect to the PDU controller. Similarly, the specification of the remaining PCs is straightforward and therefore omitted from the text, but it is available in B. The specification of the PCs are identical for all push design variants.

The specification of the ControlPC is straightforward. It includes five states. In any state the ControlPC can receive a number of legal and illegal stimuli events.

Note that, when the ControlPC is in the StartingUp state, it can send (or push) the callback event sICR_PC_CB(started) to the PDU and then transits to the Operational state. Similarly, when the ControlPC is in the Operational state, it can send the sICR_PC_CB(controlPowerOff) callback event to the PDU, as a request to power off the entire system.
The design of the PDU controller

There are mainly four alternative models for the PDU designs that incorporate the push strategy. The details of each of them are introduced below.

The asynchronous PDU controller In this variant the PDU controller communicates with the PCs synchronously and sequentially one-by-one, but the PCs communicate with the PDU asynchronously. The PDU includes a queue to store incoming callback events from the PCs.

The first issue we encountered when verifying this variant was the queue size and the large number of interleaving caused by the queue and the external commands. The PCs can quickly send callback events to the queue leading to filling-up a queue of any arbitrary size. External commands can arrive while there are still unprocessed callbacks in the queue, hence verification was initially not doable.

Therefore, we had to limit the behavior of the PCs such that having more than one similar callback at a time in the queue is prohibited. Furthermore, we give any callback event a priority to be processed by the PDU over any external user command, so the queue has to be emptied first.

Below we introduce a part of the design specification, demonstrating only the PDU Off stable state and the StartingUp_CR_PC transiting state. The entire specification can be found in Appendix B.
the *StartingUp_CR_PC* state. The other remaining summands of the state specify that any (late) callback events received in the state are consumed.

When the PDU is in the *StartingUp_CR_PC*, it can process a number of external commands and internal callbacks. All callbacks are ignored except those originating from the ControlPC. When the PDU knows that the ControlPC is ready and operational by receiving the $rICR_{PC, CB}(started)$ from the queue, it sends an indication to the user that the system is in the *SystemStandby* state. Receiving a request to power off the system from the ControlPC is illegal since the ControlPC has to start first, see the summand that corresponds to the $rICR_{PC, CB}(controlPowerOff)$ stimulus event.

The specification of the PDU controller includes a number of data parameters used for remembering the status of the PCs. For example, the $startedPc$ data parameter is of type natural number and is used to count how many normal PCs have started.

**The asynchronous PDU controller with global synchronous communication**  The model of this variant is almost identical to the previous model, except that the fourth guideline is used. We noticed that powering on/off the PCs can be modeled using multi-actions. That is, instead of modeling this behavior by sending the *powerOn* or *powerOff* events to the PCs sequentially, all PCs engage into one big action, denoting that the event occurs at the same time for all PCs.

To clarify the concept, consider the following examples. The following Handler process communicates with the PDU (via the *rcommandhandler* and *srelease* actions) and the PCs (via the *sIPC* action), where all communications are done sequentially until completion. This process is used in the specification of the asynchronous push model addressed earlier.

```
proc Handler =
  sum c:Command . rcommandhandler(c) | sIPC(1,c) .
  sIPC(2,c) . sIPC(3,c) .sIPC(4,c) . srelease | sIPC(5,c) . Handler
```

Obviously, this process results in five successive states, with the possibility of interleaving with other processes.

On the other hand, the following Handler process describes the use of multi-actions, used for this design variant. All communications with PCs are done in one step.

```
proc Handler =
  sum c:Command . rcommandhandler(c) | sIPC(1,c) | sIPC(2,c) | sIPC(3,c) | sIPC(4,c) | sIPC(5,c) . Handler
```

Clearly, this process results in a single state.

Since the number of states are reduced to a single state, the entire state space can also be reduced, taking into account the reduced interleaving. The complete specification of this model is listed in C.

**The synchronous PDU controller**  In this variant all interactions between the PDU and the PCs are synchronous. In contrast with the previous variants, the PDU does not include any queue, and all received callbacks from the PCs are processed synchronously. Still, all PCs inform (or push) the PDU upon the changes of their states, but in a synchronous manner.

The specification of this variant is listed in D. The specification is similar to the asynchronous variant except that the queue placed between the PDU and the PCs is removed.

**The synchronous PDU controller with global synchronous communication**  Here, the model of synchronous PDU controller above is adapted, such that powering on/off PCs is accomplished by multi-actions. The detail of using multi-actions is previously described for the asynchronous controller with global synchronous communication variant, and hence is omitted here. The complete specification of this variant is introduced in E.
7 Implementing the PDU controller using the poll strategy

In this section we present a model that describes the implementation of the PDU controller using a polling strategy. We used the first guideline to accomplish this model. The PDU controller polls the PCs to acquire their states. Figure 6 visualizes an example of polling used for designing the controller.

![Diagram of polling strategy](image)

Figure 6: Example of a scenario where polling is used

7.1 The external behavior of the PCs

Before describing the design of the controller, we first need to describe the external behavior of the PCs. Below, a fragment of the mCRL2 specification related to the external behavior of the ControlPC is described. The specification of the GeoPC and the normal PCs are straightforward and almost identical to this specification.

```
proc ControlPC(s:PCState) =
  (s==PC_Off) ->
    (rICR_PC(powerOn) . ControlPC(PC_On) +
     rICR_PC(powerOff) . Illegal . delta +
     rICR_PC(queryCRPCStatus) . Illegal . delta +
     rICR_PC(queryCRPCPowerOffFlag) . Illegal . delta +
     rICR_PC_Broadcast(restart) . Illegal . delta +
     rICR_PC_Broadcast(shutdown) . Illegal . delta)
    +
  (s==PC_On) ->
    (rICR_PC(powerOn) . Illegal . delta +
     rICR_PC(powerOff) . ControlPC(PC_Off) +
     rICR_PC(queryCRPCStatus) . sICR_PCrVal(IsOperational) . ControlPC(PC_On) +
     rICR_PC(queryCRPCPowerOffFlag) . sICR_PCrVal(IsNotOperational) . ControlPC(PC_0n) +
     rICR_PC(queryCRPCPowerOffFlag) . sICR_PCrVal(IsOn) . ControlPC(PC_On) +
     rICR_PC(queryCRPCPowerOffFlag) . sICR_PCrVal(IsOff) . ControlPC(PC_0n)+
     rICR_PC_Broadcast(restart) . ControlPC(PC_On) +
     rICR_PC_Broadcast(shutdown) . ControlPC(OS_Shutdown)
    )
  +
  (s==OS_Shutdown) ->
    (rICR_PC(powerOn) . Illegal . delta +
     rICR_PC(powerOff) . ControlPC(PC_Off) +
     rICR_PC(queryCRPCStatus) . Illegal . delta +
     rICR_PC(queryCRPCPowerOffFlag) . sICR_PCrVal(IsOn) . ControlPC(OS_Shutdown) +
     rICR_PC_Broadcast(restart) . ControlPC(PC_OS_Shutdown) +
     rICR_PC_Broadcast(shutdown) . ControlPC(OS_Shutdown)
  )
);```

As can be seen from the specification, the ControlPC has three main states:

- **PC_Off**: the PC is off, which means that the tap is switched off;
- **PC\_On**: the PC is on, which means that the tap is switched on, but the applications on the PC can be operational or not; and
- **OS\_Shutdown**: the tap is on but the OS and the applications are shut down.

Per state it is defined which calls are allowed to be issued by the design of the PDU controller, and which of them are illegal. Every query (or poll) method has a return value that is immediately sent back to the PDU.

When the ControlPC is powered on, it can receive a number of signals by polling. The ControlPC non-deterministically replies to these signals indicating its current state: for example, observe the summands with `queryCRPCStatus` and `queryCRPCPowerOffFlag` calls which non-deterministically return a value in the `PC\_On` state.

### 7.2 The design of the PDU controller

The PDU controller design has to adhere to the external specification of the PDU on the one hand, and to correctly use the specifications of the PCs on the other hand. To implement a polling mechanism, the PDU utilizes internal timers to stimulate the PDU to poll status of the PCs in certain states. As we will see shortly, the fourth guideline is employed to abstract from concrete data values of the timer. For example, we abstract from the progress of timer values in milliseconds by a single event denoting the expiration of the time.

Moreover, the third guideline is used for modeling the start-up behavior of the system. Compared to the push model the PDU sequentially polls information about the state of the PCs, when it is needed. The PDU does not expect any spontaneous information to be pushed by the PCs. The complete specification of this variant can be found in F.

Below we introduce a fragment of the controller design specification, related to `PDU\_Off`, `StartingUpCrPC` and `WaitingCRPCReply` states.

```plaintext
proc PDU\_State\_Machine(s:PDUState,cRPCstarted,geoPCstarted,geoPressed:Bool,state:PDUState) =
  (s==PDU\_Off) ->
    ( IPDU(PDUswitchOn) . sICR\_PC(powerOn) . IndicationCB(startingUp) .
      PDU\_State\_Machine(StartingUpCrPC,false,false,false,none) ) +
  (s==StartingUpCrPC) ->
    ( IPDU(PDUswitchOff) . sICR\_PC(powerOff) .
      PDU\_State\_Machine(PDU\_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
      IPDU(powerOn) . PDU\_State\_Machine(StartingUpCrPC,cRPCstarted,geoPCstarted,geoPressed,state) +
      IPDU(powerOff) . PDU\_State\_Machine(StartingUpCrPC,cRPCstarted,geoPCstarted,geoPressed,state) +
      IPDU(forcedPowerOff) . sICR\_PC(powerOff) . IndicationCB(off) .
      PDU\_State\_Machine(System\_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
      IPDU(emergencyOff) . sICR\_PC(powerOff) . IndicationCB(off) .
      PDU\_State\_Machine(Emergency\_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
      IPDUTimer(pollPC) . sICR\_PC(queryCRPCStatus) .
      PDU\_State\_Machine(WaitingCRPCReply,cRPCstarted,geoPCstarted,geoPressed,state) ) +
  ...;
```

The fragment describes that when the system is switched on in the `PDU\_Off` state, the ControlPC is powered on, the user gets an indication that the system is starting up, and the PDU transits to the `StartingUpCrPC` state. As can be inferred from the specification, the `StartingUpCrPC` state is used to not only monitor the progress of starting up the ControlPC, but also to react upon the external requests from users.

Then, when the PDU is stimulated by the timer via the `pollPC` signal, the PDU requests the state of the ControlPC by sending the `queryCRPCStatus` signal and transits to the `WaitingCRPCReply` state, waiting a response from the ControlPC. As specified in the external behavior of the ControlPC, either `IsOperational` or `IsNotOperational` signals are returned to the PDU. Depending on the return value, the PDU transits back to `StartingUpCrPC` (and hence can query the status...
of the ControlPC again), or gives an indication that the system is in standby and transits to the System Standby state.

Similarly, when the system is in the System Standby stable state and the PowerOn button is pressed, the PDU transits to the StartingUpAllPCs state where all other PCs are checked, in the same manner of checking the status of the ControlPC described above.

\[
\text{(s==WaitingPC1statusReply) -> (}
\begin{align*}
&\text{rIPCVal(1,IsOperational)} . \text{sIPC(2,queryPCstatus)} . \\
&\text{PDU State Machine(WaitingPCstatusReply,cRPCstarted,geoPCstarted,geoPressed,state) +} \\
&\text{rIPCVal(1,IsNotOperational)} . \\
&\text{PDU State Machine(StartingUpAllPCs,cRPCstarted,geoPCstarted,geoPressed,state)} \\
&\text{)} + \\
&\text{(s==WaitingPC5statusReply) -> (}
\begin{align*}
&\text{(!geoPressed) -> rIPCVal(5,IsOperational)} . \text{IndicationCB(systemOn)} . \\
&\text{PDU State Machine(System_On,cRPCstarted,geoPCstarted,geoPressed,state) +} \\
&\text{(geoPressed) -> rIPCVal(5,IsOperational)} . \text{IndicationCB(geoStop)} . \\
&\text{PDU State Machine(Geo_Stop,cRPCstarted,geoPCstarted,geoPressed,state) +} \\
&\text{rIPCVal(5,IsNotOperational)} . \\
&\text{PDU State Machine(StartingUpAllPCs,cRPCstarted,geoPCstarted,geoPressed,state)} \\
&\text{)} \\
&\text{...}
\end{align*}
\]

That is, the first PC is checked if it is operational or not. If the first PC is not operational, the system can transit back to the StartingUpAllPCs state; see for example the specification of the WaitingPC1statusReply state above. If the first PC is operational, then the second PC is checked, and so on until all PCs are operational. When the last PC is operational, an indication is sent to the user, and then the PDU moves to the System On state, see the WaitingPC5statusReply state.

During starting up of all PCs, the PDU queries the GeoPC and the ControlPC to check the status of whether any of the Stop buttons has been pressed or if the user needs to power off the entire system. If these flags are on, on the respective PCs, the PDU immediately switches off the taps supply the movable part or starts to power off the entire system. The PDU remembers the status of the Stop button, and therefore, when the last PC is operational, the PDU transits to System On or Geo Stop stable states.

The poll controller with global synchronous communication In combination with the first guideline, we use guideline 2 to model the instantaneous powering on or off the PCs. The same global synchronous communication concept used for the Handler process of the push model is also used here. We refer to G for the entire specification of this model.

8 Results of the experiments

After the specification of all models were created using the mCRL2 description language, we started the verification tasks. We used the mCRL2 tool set (July 2011 release) for performing verification and state space generation on a Unix-based server machine (4 × 2.5 Ghz processor and 46 GB RAM). The generated state spaces of all models were further analyzed using CADP (June 2011 beta release) for checking deadlocks, livelocks, illegals and proving refinements of designs against the external specification.

Table 4 depicts the activities performed throughout this work together with the tools used to accomplish each of them. The ‘✓’ mark indicates a feature supported by the tool and being used in this work, ‘−’ denotes that the feature is supported by the tool but is not being used in this work, and ‘×’ indicates that the tool does not support the feature. As can be seen from the table, the formal specification using CADP is skipped since we used the mCRL2 for state space generation. The state space was analyzed later using both mCRL2 and CADP. We also translated the mCRL2 models to CSP and used FDR for state space generation. FDR was used to verify refinements under traces, failures and failures-divergence models, of which the last two are not supported by both mCRL2 and CADP. When the state space of each model has been generated, branching bisimulation reduction was applied after all internal events not visible on the external specification are hidden, to facilitate the verification and refinement tasks.
The three tools were used for searching for occurrences of deadlocks and illegals. All tools provided the same result, namely all models are deadlock and illegal free. After we hid all events except those exposed in the external specification, we checked for the occurrences of livelocks. Checking livelocks merely was accomplished using FDR (FDR2 2.91 academic use release). The reason of choosing FDR over other tools is that FDR provides readable, easy to analyze, counterexamples in case livelocks exist. The mCRL2 for example can report a sequence that leads to a cycle of \(\text{tau}\) events, but one can hardly deduce the corresponding original actions that form the cycle. The same applies for CADP.

We encountered a similar issue when trying to prove refinement of designs against the external behavior using both mCRL2 and CADP. The tools can easily find counterexamples when a refinement check is violated, under the refinement models they support. But, the generated counterexamples were hard to read since all original internal actions were permanently replaced by the hidden action \(\text{tau}\). By using mCRL2 and CADP, we spent extra time analyzing the counterexamples and to ‘guess’ the correct original events correspond to the hidden events by matching the sequence of \(\text{tau}\)’s on the original system. This indeed caused more efforts and time to be spent for modeling and verification since we did not efficiently know whether the design or the external specification was incorrect. Notable is that knowing the original actions correspond to the hidden action \(\text{tau}\) when checking refinements was straightforward in CSP/FDR.

However, when we attempted to verify an initial model of the push design using FDR, the tool quickly crashed during the compilation phase. The reason is that the model initially implements a list to store the started PCs, see the \(\text{startedPc}\) data parameter in the push model introduced earlier. The controller needs this list during the start-up of the system in order to know that all PCs are fully operational before moving to the \(\text{System} \_ \text{On}\) state. It seems that having such a list in our model caused FDR to crash, and thus when replacing the list by a counter, the issue was solved indeed. Notable is that mCRL2 dealt with both types of push models that include either a list or a counter of started PCs effectively. The last four refinement checks were performed using CADP, which was the only tool supporting them.

In table 5 we summarize the end result of checking refinements of designs, under a number of refinement models. The table is self-explainable. All designs refine the external specification under all refinement models, which means that all designs provide the expected behavior to external users of the system according to the predefined external specification. The only exception is the refinement of the poll models under the failures-divergences model, which fails due to the presence of a livelock.

The livelock exists in the poll models since internal \(\text{tau}\) loops can easily be formed, see Figure 7 for a livelock scenario. For example, in case the ControlPC is not operational, the PDU controller will query it again. This can continue forever, unless the ControlPC becomes operational. But, since the external users can still issue external commands even if the ControlPC is not operational, we consider this livelock to be rather benign, and indeed the livelock represents a desired and
Table 5: Results of checking refinements of designs against the external behavior

<table>
<thead>
<tr>
<th>Model</th>
<th>Weak-traces</th>
<th>Failures</th>
<th>Fail. Diverg.</th>
<th>observational</th>
<th>safety</th>
<th>Tau*</th>
<th>branching</th>
</tr>
</thead>
<tbody>
<tr>
<td>Async. Push</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Async. Push Global Sync.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Sync Push</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Sync Push Global sync.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Poll</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Poll Global sync.</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Note that all designs are deadlock, livelock and illegal free except the poll designs, which are not livelock free due to the above mentioned reason.

Figure 7: A divergence example

Table 6: State spaces of all models

<table>
<thead>
<tr>
<th>Model</th>
<th>States</th>
<th>Transitions</th>
<th>BB</th>
<th>BBDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>External specification</td>
<td>15</td>
<td>53</td>
<td>13</td>
<td>46</td>
</tr>
<tr>
<td>Async. Push</td>
<td>78,088,550</td>
<td>122,354,296</td>
<td>47</td>
<td>173</td>
</tr>
<tr>
<td>Async. Push Global Sync.</td>
<td>44,866,381</td>
<td>75,945,810</td>
<td>47</td>
<td>173</td>
</tr>
<tr>
<td>Push sync</td>
<td>6,318</td>
<td>8,486</td>
<td>23</td>
<td>111</td>
</tr>
<tr>
<td>Push sync global sync</td>
<td>3,832</td>
<td>6,000</td>
<td>23</td>
<td>111</td>
</tr>
<tr>
<td>Poll</td>
<td>953</td>
<td>1,367</td>
<td>14</td>
<td>54</td>
</tr>
<tr>
<td>Poll global sync</td>
<td>608</td>
<td>1,022</td>
<td>14</td>
<td>54</td>
</tr>
</tbody>
</table>

The last table sums up the statistical data related to the size of generated state spaces. The second and third columns shows the number of generated states and transitions for the entire state spaces. The branching-bisimulation (BB) columns depict the number of resulting states and transitions after the branching-bisimulation reduction was applied on the original state space, while those resulting from branching-bisimulation compression with divergence preserving (BBDP) are depicted in the last columns.

1In fact there are additional services deployed on a number of PCs for monitoring the status of PCs. If they detect that there is some PC has failed to start, they try to start it again using its baseboard management control (BMC) via its intelligent platform management interface (IPMI), through the Ethernet network. The PDU team is not responsible of implementing these services.
The difference between the number of transitions of the poll models after compression using BB and BBDP indicates that the poll model includes divergences. A divergence scenario of the poll model was discussed earlier.

As can be seen from the table, the poll variants appear to be better than others, with only 953 and 608 states. They show also fewer states after compression. This favorably compares to the asynchronous push model which includes 78,088,550 states. Therefore, it seems that extending the asynchronous push model further with extra details may limit the verification process, unless the design of the PDU controller is decomposed into a number of smaller components verified in isolation, or on-the-fly reduction techniques are used for circumventing a foreseen state space explosion.

Finally, the above results indicate that different design styles can substantially influence the number of states of the modeled systems. This confirms that these design styles are effective in practice. Although more experiments need to be done with these different design styles, we are strengthened in our believe that these styles are very important and designers should be actively aware of such strategies if they want to design verifiable systems.

References


A External behaviour

sort State = struct PDU_Off | System_StandBy | System_On | Emergency_Off | System_Off | Geo_Stop |
Off | StartingUp | StartingUpCrPC | StartingUpAllPCs;
Command = struct PDUswitchOn | PDUswitchOff | powerOn | powerOff | forcedPowerOff | emergencyOff;
IndicationMsg = struct startingUp | off | systemOn | systemStandby | geoStop;
act IPDU:Command;
act IndicationCB: IndicationMsg;
proc ExtSpec(s:State)=
(s==PDU_Off) ->
   ( IPDU(PDUswitchOn) . IndicationCB(startingUp) . ExtSpec(StartingUpCrPC) ) +
(s==System_StandBy) ->
   ( IPDU(PDUswitchOff) . ExtSpec(PDU_Off) +
     IPDU(powerOn). IndicationCB(startingUp) . ExtSpec(StartingUpAllPCs)+
     IPDU(powerOff) . ExtSpec(System_StandBy) +
     IPDU(forcedPowerOff) .
     IndicationCB(off) . ExtSpec(System_Off) +
     IPDU(emergencyOff) .
     IndicationCB(off) . ExtSpec(Emergency_Off) +
     tau . IndicationCB(off) . ExtSpec(System_Off) ) +
(s==System_On) ->
   ( IPDU(PDUswitchOff) . ExtSpec(PDU_Off) +
     IPDU(powerOn) . IndicationCB(systemStandby) . ExtSpec(System_StandBy)+
     IPDU(powerOff) .
     IndicationCB(systemStandby) . ExtSpec(System_StandBy) +
     IPDU(forcedPowerOff) .
     IndicationCB(off) . ExtSpec(System_Off) +
     IPDU(emergencyOff) .
     IndicationCB(off) . ExtSpec(Emergency_Off) +
     tau . IndicationCB(off) . ExtSpec(System_Off) +
     tau . IndicationCB(geoStop). ExtSpec(geoStop) . ExtSpec(Geo_Stop) ) +
(s==StartingUpAllPCs) ->
   ( IPDU(PDUswitchOff) . ExtSpec(PDU_Off) +
     IPDU(powerOn). ExtSpec(StartingUpAllPCs) +
     IPDU(powerOff) . IndicationCB(systemStandby) . ExtSpec(System_StandBy)+
     IPDU(powerOff) .
     ExtSpec(StartingUpCrPC)+
     IPDU(forcedPowerOff) .
     IndicationCB(geoStop) . ExtSpec(System_Off) +
     tau . IndicationCB(geoStop) . ExtSpec(geoStop) ) +
(s==Geo_Stop) ->
   ( IPDU(PDUswitchOff) . ExtSpec(PDU_Off) +
     IPDU(powerOn). IndicationCB(systemOn) . ExtSpec(System_On) +
     IPDU(powerOff). IndicationCB(systemStandby).ExtSpec(System_StandBy)+
     IPDU(forcedPowerOff) .
     IndicationCB(systemOn) . ExtSpec(System_Off) +
     IPDU(emergencyOff) .
     IndicationCB(systemOn) . ExtSpec(System_Off) +
     tau . IndicationCB(systemOn) . ExtSpec(System_Off) +
     tau . IndicationCB(geoStop). ExtSpec(geoStop) ) +
(s==System_Off) ->
   ( IPDU(PDUswitchOff) . ExtSpec(PDU_Off) +
     IPDU(powerOn). IndicationCB(startingUp) . ExtSpec(StartingUpAllPCs)+
     IPDU(powerOff) . ExtSpec(System_Off) +
     IPDU(forcedPowerOff) .
     ExtSpec(System_Off) +
     IPDU(emergencyOff) .
     ExtSpec(System_Off) +
     tau . IndicationCB(geoStop) . ExtSpec(System_Off) ) +
(s==Emergency_Off) ->
   ( IPDU(PDUswitchOff) . ExtSpec(PDU_Off) +
     IPDU(powerOn). IndicationCB(startingUp) . ExtSpec(StartingUpAllPCs) +
     IPDU(powerOff) . ExtSpec(System_Off) +
     IPDU(forcedPowerOff) .
     ExtSpec(System_Off) +
     IPDU(emergencyOff) .
     ExtSpec(System_Off) +
     IPDU(emergencyOff) .
     ExtSpec(System_Off) ) +
(s==StartingUpCrPC) ->
   ( IPDU(PDUswitchOff) . ExtSpec(PDU_Off) +
B  Asynchronous push model

sort Command = struct PDUswitchOn | PDUswitchOff | powerOn | powerOff | forcedPowerOff
| emergencyOff | onPressed;

Bmsg = struct restart | shutdown ;

PCsCallbacks = struct controlPowerOff | stop | started ;

PCState = struct PC_Off | Operational | WaitingShutdown | StartingUp | GS_Shutdown | StopPressed ;

PDUState = struct PDU_Off | StartingUp_CR_PC | StartingUpAllPcs | SystemStandby | System_On
| Emergency_Off | System_Off | Geo_Stop ;

IndicationMsg = struct startingUp | off | systemOn | systemStandby | geoStop ;

PCs = struct CRPC | GeoPC | NormalPC ;

CBMsg = struct msg(pc:PCs,id:Pos,cb:PCsCallbacks);

act Illegal, release, rrelease, release;

sIPDU, rIPDU, IPDU,
sICR_PC, rICR_PC, ICR_PC,
sGeoPC, rGeoPC, IGeoPC,
scommandhandler, rcommandhandler, commandhandler : Command;

sIPC, rIPC, IPC : Pos # Command;  
sICR_PC_Broadcast, rICR_PC_Broadcast, ICR_PC_Broadcast,
sGeoPC_Broadcast, rGeoPC_Broadcast, IGeoPC_Broadcast,
sMeghandler, rMeghandler, Meghandler : Bmsg;

sIPC_Broadcast, rIPC_Broadcast, IPC_Broadcast : Pos # Command;

sICR_PC_CB, rICR_PC_CBout, ICR_PC_CBout, sICR_PC_CB, rICR_PC_CBin, ICR_PC_CBin,
sGeoPC_CB, sGeoPC_CBout, IGeoPC_CBout, sGeoPC_CB, rGeoPC_CBin, IGeoPC_CBin : PCsCallbacks;

rIPC_CB, sIPC_CBout, IPC_CBout, sIPC_CB, rIPC_CBin, IPC_CBin : Pos # PCsCallbacks;

IndicationCB : IndicationMsg;

proc ControlPC(s:PCState) = (  
(s==PC_Off) ->
(rICR_PC(powerOn)).ControlPC(StartingUp) +
(rICR_PC(powerOff)).Illegal . delta +
(rICR_PC_Broadcast(restart)).Illegal . delta +
(rICR_PC_Broadcast(shutdown)).Illegal . delta
)
+  
(s==Operational) ->
(rICR_PC(powerOn)).Illegal . delta +
(rICR_PC(powerOff)).ControlPC(PC_Off) +
sICR_PC_CB(controlPowerOff).ControlPC(WaitingShutdown) +
rICR_PC_Broadcast(restart).ControlPC(StartingUp) +
rICR_PC_Broadcast(shutdown).ControlPC(GS_Shutdown)
)+

(s==WaitingShutdown) ->
(rICR_PC(powerOn)).Illegal . delta +
(rICR_PC(powerOff)).ControlPC(PC_Off) +
rICR_PC_Broadcast(restart).ControlPC(StartingUp) +
rICR_PC_Broadcast(shutdown).ControlPC(GS_Shutdown)
)+

(s==GS_Shutdown) ->
(rICR_PC(powerOn)).Illegal . delta +
(rICR_PC(powerOff)).ControlPC(PC_Off) +
rICR_PC_Broadcast(restart).Illegal . delta +
(rICR_PC_Broadcast(shutdown)).Illegal . delta
)+
proc GeoPC(s:PCState) = (  
  (s==PC_Off) ->  
  ( rIGeoPC_Broadcast(shutdown) . Illegal . delta +  
    rIGeoPC(powerOn) . GeoPC(StartingUp) +  
    rIGeoPC(onPressed) . Illegal . delta +  
    rIGeoPC(powerOff) . Illegal . delta +  
    rIGeoPC_Broadcast(restart) . Illegal . delta +  
  )+  
  (s==Operational) ->  
  ( rIGeoPC(powerOn) . Illegal . delta +  
    rIGeoPC(powerOff) . GeoPC(PC_Off) +  
    rIGeoPC(onPressed) . GeoPC(Operational) + % it can happen if the restart is issued before  
    rIGeoPC_Broadcast(shutdown) . GeoPC(OS_Shutdown) +  
    rIGeoPC_Broadcast(restart) . GeoPC(StartingUp) +  
    sIGeoPC_CB(stop) . GeoPC(StopPressed)  
  )+  
  (s==StopPressed) ->  
  ( rIGeoPC(powerOn) . Illegal . delta +  
    rIGeoPC(powerOff) . GeoPC(PC_Off) +  
    rIGeoPC_Broadcast(shutdown) . GeoPC(OS_Shutdown) +  
    rIGeoPC_Broadcast(restart) . GeoPC(StartingUp) +  
    rIGeoPC(onPressed) . GeoPC(Operational)  
  )+  
  (s==OS_Shutdown) ->  
  ( rIGeoPC(powerOn) . Illegal . delta +  
    rIGeoPC(powerOff) . GeoPC(PC_Off) +  
    rIGeoPC(onPressed) . Illegal . delta +  
    rIGeoPC_Broadcast(shutdown) . Illegal . delta +  
    rIGeoPC_Broadcast(restart) . Illegal . delta +  
  )+  
  (s==StartingUp) ->  
  ( rIGeoPC(powerOn) . Illegal . delta +  
    rIGeoPC(powerOff) . GeoPC(PC_Off) +  
    rIGeoPC_Broadcast(shutdown) . GeoPC(StartingUp) +  
    rIGeoPC_Broadcast(restart) . Illegal . delta +  
  )+  
  (s==StopPressed) ->  
  ( rIGeoPC(powerOn) . Illegal . delta +  
    rIGeoPC(powerOff) . GeoPC(PC_Off) +  
    rIGeoPC_Broadcast(shutdown) . GeoPC(OS_Shutdown) +  
    rIGeoPC_Broadcast(restart) . GeoPC(StartingUp) +  
    rIGeoPC(onPressed) . GeoPC(Operational)  
  )+  
  (s==OS_Shutdown) ->  
  ( rIGeoPC(powerOn) . Illegal . delta +  
    rIGeoPC(powerOff) . GeoPC(PC_Off) +  
    rIGeoPC_Broadcast(shutdown) . GeoPC(OS_Shutdown) +  
    rIGeoPC_Broadcast(restart) . GeoPC(StartingUp) +  
    sIGeoPC_CB(started) . GeoPC(Operational)  
  )+  
):  
):  
proc NormalPC(id:Pos, s:PCState) = (  
  (s==PC_Off) ->  
  ( rIPC(id,powerOn) . NormalPC(id,StartingUp) +  
    rIPC(id,powerOff) . Illegal . delta +  
    rIPC_Broadcast(id,shutdown) . Illegal . delta +  
    rIPC_Broadcast(id,restart) . Illegal . delta +  
  )+  
  (s==Operational) ->  
  ( rIPC(id,powerOn) . Illegal . delta +  
    rIPC(id,powerOff) . NormalPC(id,PC_Off) +  
    rIPC_Broadcast(id,shutdown) . NormalPC(id,OS_Shutdown) +  
    rIPC_Broadcast(id,restart) . NormalPC(id,StartingUp)  
  )+  
  (s==OS_Shutdown) ->  
  ( rIPC(id,powerOn) . Illegal . delta +  
    rIPC(id,powerOff) . NormalPC(id,PC_Off) +  
    rIPC_Broadcast(id,shutdown) . Illegal . delta +  
    rIPC_Broadcast(id,restart) . Illegal . delta +  
  )+  
):
% the PDU design

proc PDU_State_Machine(s:PDUState,geopcOn,crpcOn,geoPressed:Bool,startedPc:Nat) = (  
(s==PDU_Off) ->  
  ( rIPDU(PDUswitchOn) . sICR_PC(powerOn) . sICR_PC_Broadcast(shutdown) . sIGeoPC_Broadcast(shutdown) . sMsghandler(shutdown) . rrelease . sIGeoPC(powerOff) . scommandhandler(powerOff) . rrelease . IndicationCB(systemStandby) . PDU_State_Machine(SystemStandby,geopcOn,crpcOn,geoPressed,startedPc) +  
  rIPDU(poweredOn) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +  
  rIPDU(powerOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +  
  rIPDU(forcedPowerOff) . sICR_PC_Broadcast(shutdown) . sIGeoPC_Broadcast(shutdown) . sMsghandler(shutdown) . rrelease . sIGeoPC(powerOff) . scommandhandler(powerOff) . rrelease . IndicationCB(systemStandby) . PDU_State_Machine(SystemStandby,geopcOn,crpcOn,geoPressed,startedPc) +  
  sum id:Pos . rIPC_CB(id,started) . PDU_State_Machine(SystemStandby,geopcOn,crpcOn,geoPressed,startedPc) +  
  sIPC_CB(id,started) . Illegal . delta +  
  sIPC_CB(id,started) . Illegal . delta +  
  )  
(s==StartingUp) ->  
  ( rIPC(id,powerOn) . Illegal . delta +  
  rIPC(id,powerOff) . NormalPC(id,PC_Off) +  
  rIPC_Broadcast(id,shutdown) . NormalPC(id,StartingUp) +  
  rIPC_Broadcast(id,restart) . Illegal . delta +  
  sIPC_CB(id,started) . NormalPC(id,Operational) +  
  )  
(s==StartingUp_CR_PC) ->  
  ( rIPDU(PDUswitchOff) . sICR_PC(powerOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +  
  rIPDU(powerOn) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +  
  rIPDU(powerOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +  
  rIPDU(forcedPowerOff) . sICR_PC(powerOff) . sICR_PC_Broadcast(shutdown) . sIGeoPC_Broadcast(shutdown) . sMsghandler(shutdown) . rrelease . sIGeoPC(powerOff) . scommandhandler(powerOff) . rrelease . IndicationCB(systemStandby) . PDU_State_Machine(SystemStandby,geopcOn,crpcOn,geoPressed,startedPc) +  
  sum id:Pos . rIPC_CB(id,started) . PDU_State_Machine(SystemStandby,geopcOn,crpcOn,geoPressed,startedPc) +  
  rIPC_Broadcast(id,shutdown) . Illegal . delta +  
  rIPC_Broadcast(id,restart) . Illegal . delta +  
  )  
(s==SystemStandby) ->  
  ( rIPDU(PDUswitchOff) . sICR_PC(powerOff) . sICR_PC_Broadcast(shutdown) . sIGeoPC_Broadcast(shutdown) . sMsghandler(shutdown) . rrelease . sIGeoPC(powerOff) . scommandhandler(powerOff) . rrelease . IndicationCB(systemStandby) . PDU_State_Machine(SystemStandby,geopcOn,crpcOn,geoPressed,startedPc) +  
  rIPDU(poweredOn) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +  
  rIPDU(powerOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +  
  rIPDU(forcedPowerOff) . sICR_PC(powerOff) . IndicationCB(powerOff) . rrelease . sICR_PC_Broadcast(shutdown) . sIGeoPC_Broadcast(shutdown) . sMsghandler(shutdown) . rrelease . sIGeoPC(powerOff) . scommandhandler(powerOff) . rrelease . IndicationCB(systemStandby) . PDU_State_Machine(SystemStandby,geopcOn,crpcOn,geoPressed,startedPc) +  
  sum id:Pos . rIPC_CB(id,started) . PDU_State_Machine(SystemStandby,geopcOn,crpcOn,geoPressed,startedPc) +  
  rIPC_Broadcast(id,shutdown) . Illegal . delta +  
  rIPC_Broadcast(id,restart) . Illegal . delta +  
  )  
(s==System_On) ->  
  ( rIPDU(PDUswitchOff) . sICR_PC(powerOff) . sICR_PC_Broadcast(shutdown) . sIGeoPC_Broadcast(shutdown) . sMsghandler(shutdown) . rrelease . sICR_PC(powerOff) . sICR_PC_Broadcast(shutdown) . sIGeoPC_Broadcast(shutdown) . sMsghandler(shutdown) . rrelease . sIGeoPC(powerOff) . scommandhandler(powerOff) . rrelease . IndicationCB(systemStandby) . PDU_State_Machine(SystemStandby,geopcOn,crpcOn,geoPressed,startedPc) +  
  rIPDU(poweredOn) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +  
  rIPDU(powerOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +  
  rIPDU(powerOn) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +  
  rIPDU(forcedPowerOff) . sICR_PC(powerOff) . IndicationCB(powerOff) . rrelease . sICR_PC_Broadcast(shutdown) . sIGeoPC_Broadcast(shutdown) . sMsghandler(shutdown) . rrelease . sIGeoPC(powerOff) . scommandhandler(powerOff) . rrelease . IndicationCB(systemStandby) . PDU_State_Machine(SystemStandby,geopcOn,crpcOn,geoPressed,startedPc) +  
  sum id:Pos . rIPC_CB(id,started) . PDU_State_Machine(SystemStandby,geopcOn,crpcOn,geoPressed,startedPc) +  
  rIPC_Broadcast(id,shutdown) . Illegal . delta +  
  rIPC_Broadcast(id,restart) . Illegal . delta +  
  )

)}
rIPDU(forcedPowerOff) . sICR_PC(powerOff) . sIGeoPC(powerOff) . scommandhandler(powerOff) .
   release . IndicationCB(off) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
   rIPDU(emergencyOff) . sICR_PC(powerOff) . sIGeoPC(powerOff) . scommandhandler(powerOff) .
   release . IndicationCB(off) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
rICR_PC_CB(controlPowerOff) . sICR_PC_Broadcast(shutdown) . sIGeoPC_Broadcast(shutdown) .
   sMsghandler(shutdown) . release . sICR_PC(powerOn) . sIGeoPC(powerOn) .
   scommandhandler(powerOn) . release . IndicationCB(off) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
rIGeoPC_CB(stop) . PDU_State_Machine(Geo_Stop,geopcOn,crpcOn,geoPressed,startedPc) +
   rICR_PC_CB(started) . Illegal . delta +
   rIGeoPC_CB(started) . Illegal . delta +
sum id:Pos . rIPC_CB(id,started) . Illegal . delta
)

(s==Emergency_Off) ->
   (rIPDU(PDUswitchOff) . PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) +
   rIPDU(powerOn) . sICR_PC(powerOn) . sIGeoPC(powerOn) . scommandhandler(powerOn) .
   release . IndicationCB(startingUp) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
   rIPDU(powerOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
   rIPDU(emergencyOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
rICR_PC_CB(controlPowerOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
rIGeoPC_CB(started) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
   sum id:Pos . rIPC_CB(id,started) .
   PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc)
)

(s==System_Off) ->
   (rIPDU(PDUswitchOff) . PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) +
   rIPDU(powerOn) . sICR_PC(powerOn) . sIGeoPC(powerOn) . scommandhandler(powerOn) .
   release . IndicationCB(startingUp) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
   rIPDU(powerOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
   rIPDU(emergencyOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
rICR_PC_CB(controlPowerOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
rIGeoPC_CB(started) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
   sum id:Pos . rIPC_CB(id,started) .
   PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc)
)

(s==Geo_Stop) ->
   (rIPDU(PDUswitchOff) . sICR_PC(powerOff) . sIGeoPC(powerOff) . scommandhandler(powerOff) .
   release . PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) +
   rIPDU(powerOn) . sIGeoPC(onPressed) . IndicationCB(systemOn) .
   PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
   rIPDU(powerOff) . sIGeoPC_Broadcast(shutdown) . sMsghandler(shutdown) . release .
   sIGeoPC(powerOff) . scommandhandler(powerOff) . release . IndicationCB(systemStandby) .
   PDU_State_Machine(SystemStandby,geopcOn,crpcOn,geoPressed,startedPc) +
   rIPDU(forcedPowerOff) . sICR_PC(powerOff) . sIGeoPC(powerOff) . scommandhandler(powerOff) .
   release . IndicationCB(off) .
   PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
   rIPDU(emergencyOff) . sICR_PC(powerOff) . sIGeoPC(powerOff) . scommandhandler(powerOff) .
   release . IndicationCB(off) .
   PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
   rICR_PC_CB(controlPowerOff) . sICR_PC_Broadcast(shutdown) . sMsghandler(shutdown) . release .
   sICR_PC(powerOff) . sIGeoPC(powerOff) .
   scommandhandler(powerOff) . release . IndicationCB(off) .
   PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
   rIGeoPC_CB(stop) . PDU_State_Machine(Geo_Stop,geopcOn,crpcOn,geoPressed,startedPc) +
   rIGeoPC_CB(started) . Illegal . delta +
   rIGeoPC_CB(started) . Illegal . delta +
sum id:Pos . rIPC_CB(id,started) . Illegal . delta
(s==StartingUpAllPcs) ->

\[
\begin{align*}
\text{rIPDU}(\text{PDU\_switch\_off}) & . \text{sICR\_PC}(\text{power\_off}) & . \text{sIGeoPC}(\text{power\_off}) & . \text{scommandhandler}(\text{power\_off}) . \\
& \text{release} & . \text{PDU\_State\_Machine}(\text{PDU\_off,geopcOn,crpcOn,geoPressed,startedPc}) & + \\
\text{rIPDU}(\text{power\_on}) & . \text{PDU\_State\_Machine}(\text{StartingUpAllPcs,geopcOn,crpcOn,geoPressed,startedPc}) & + \\
& (\text{crpcOn}) & \rightarrow & \text{rIPDU}(\text{power\_off}) & . \text{sICR\_PC}(\text{power\_off}) & . \text{sIGeoPC}(\text{power\_off}) & . \text{scommandhandler}(\text{power\_off}) . \\
& \text{release} & & . \text{sGeoPC}(\text{power\_off}) & . \text{scommandhandler}(\text{power\_off}) & . \text{release}. \\
\text{PDU\_State\_Machine}(\text{StartingUp\_CR\_PC,false,crpcOn,false,0}) & + \\
\text{rIPDU}(\text{forced\_power\_off}) & . \text{sICR\_PC}(\text{power\_off}) & . \text{sIGeoPC}(\text{power\_off}) & . \text{scommandhandler}(\text{power\_off}) . \\
& \text{release} . \text{IndicationCB}(\text{system\_standby}) & . \text{PDU\_State\_Machine}(\text{System\_off,false,crpcOn,false,0}) & + \\
\text{rIPDU}(\text{emergency\_off}) & . \text{sICR\_PC}(\text{power\_off}) & . \text{sIGeoPC}(\text{power\_off}) & . \text{scommandhandler}(\text{power\_off}) . \\
& \text{release} . \text{IndicationCB}(\text{off}) & . \text{PDU\_State\_Machine}(\text{System\_off,false,crpcOn,false,0}) & + \\
\text{rIPDU}(\text{forced\_power\_off}) & . \text{sICR\_PC}(\text{power\_off}) & . \text{sIGeoPC}(\text{power\_off}) & . \text{scommandhandler}(\text{power\_off}) . \\
& \text{release} . \text{IndicationCB}(\text{off}) & . \text{PDU\_State\_Machine}(\text{System\_off,false,crpcOn,false,0}) & + \\
\text{rIPDU}(\text{forced\_power\_off}) & . \text{sICR\_PC}(\text{power\_off}) & . \text{sIGeoPC}(\text{power\_off}) & . \text{scommandhandler}(\text{power\_off}) . \\
\text{release}. \\
\end{align*}
\]
proc System=
hide({ICR_PC, ICR_PC_Broadcast, ICR_PC_CBout, ICR_PC_CBin,
IGeoPC, IGeoPC_Broadcast, IGeoPC_CBout, IGeoPC_CBin,
IPC, IPC_Broadcast, IPC_CBout, IPC_CBin,
release,
Msghandler, commandhandler
}),
allow({IPDU,
ICR_PC, ICR_PC_Broadcast, ICR_PC_CBout, ICR_PC_CBin,
IGeoPC, IGeoPC_Broadcast, IGeoPC_CBout, IGeoPC_CBin,
IPC, IPC_Broadcast, IPC_CBout, IPC_CBin,
IndicationCB,
commandhandler | IPC, Msghandler | IPC_Broadcast, release | IPC, release | IPC_Broadcast,
sIPDU | rIPDU -> IPDU,
sICR_PC | rICR_PC -> ICR_PC,
sICR_PC_Broadcast | rICR_PC_Broadcast -> ICR_PC_Broadcast,
sIGeoPC | rIGeoPC -> IGeoPC,
sIGeoPC_Broadcast | rIGeoPC_Broadcast -> IGeoPC_Broadcast,
sIPC_CBin | rIPC_CBin -> IPC_CBin,
sICR_PC_CBout | rICR_PC_CBout -> ICR_PC_CBout,
sIGeoPC_CBout | rIGeoPC_CBout -> IGeoPC_CBout,
sIPC_CB | rIPC_CB -> IPC_CBout,
sIPDU | rIPDU -> IPDU,
sICR_PC | rICR_PC -> ICR_PC,
sICR_PC_Broadcast | rICR_PC_Broadcast -> ICR_PC_Broadcast,
sIGeoPC | rIGeoPC -> IGeoPC,
sIGeoPC_Broadcast | rIGeoPC_Broadcast -> IGeoPC_Broadcast,
sIPC_CBin | rIPC_CBin -> IPC_CBin,
sICR_PC_CBout | rICR_PC_CBout -> ICR_PC_CBout,
sIGeoPC_CBout | rIGeoPC_CBout -> IGeoPC_CBout,
sIPC_CB | rIPC_CB -> IPC_CBout,
release | release -> release,
rIPC | sIPC -> IPC,
rIPC_Broadcast | sIPC_Broadcast -> IPC_Broadcast,
sMsghandler | rMsghandler -> Msghandler,
scommandhandler | rcommandhandler -> commandhandler
},
CallBackQueue([]) ||
ControlPC(PC_Off) ||
GeoPC(PC_Off) ||
NormalPC(1, PC_Off) ||
NormalPC(2, PC_Off) ||
NormalPC(3, PC_Off) ||
NormalPC(4, PC_Off) ||
NormalPC(5, PC_Off) ||
Handler ||
PDU_State_Machine(PDU_Off, false, false, false, 0))));

init System;

C Asynchronous push model with global synchronous communication

sort Command = struct PDUswitchOn | PDUswitchOff | powerOn | powerOff | forcedPowerOff |
                     emergencyOff | onPressed;
Bmsg = struct restart | shutdown ;
PcsCallbacks = struct controlPowerOff | start | started ;
PCState = struct PC_Off | Operational | WaitingShutdown | StartingUp | OS_Shutdown |
                     StopPressed; 
PDUState = struct PDU_Off | StartingUp_CR_PC | StartingUpAllPcs | SystemStandby |
                     System_On | Emergency_Off | System_Off | Geo_Stop ; 
IndicationMsg = struct startingUp | off | systemOn | systemStandby | geoStop ; 
Pcs = struct CRPC | GeoPC | NormalPC ;
sort CBMsg = struct msg(pc:Pcs, id:Pos, cb:PcsCallbacks);
act Illegal,
release, release, release;
sIPDU, rIPDU, IPDU,
sICR_PC, rICR_PC, ICR_PC,
sGeoPC, rGeoPC, IGeoPC,
scommandhandler, rcommandhandler, commandhandler : Command;
sIPC, rIPC, IPC : Pos # Command;
sICR_PC_Broadcast, rICR_PC_Broadcast, ICR_PC_Broadcast,
sGeoPC_Broadcast, rGeoPC_Broadcast, IGeoPC_Broadcast,
smeghandler, rmeghandler, Meghandler: Bmsg;
sIPC_Broadcast, rIPC_Broadcast, IPC_Broadcast : Pos # Command;
sICR_PC_CB, rICR_PC_CBout, ICR_PC_CBout, sICR_PC_CB, rICR_PC_CBin, ICR_PC_CBin,
sIGeoPC_CB, rIGeoPC_CBout, IGeoPC_CBout, sIGeoPC_CB, rIGeoPC_CBin,
IGeoPC_CBin: PCsCallbacks;
sMe PC, sIPC_CBout, IPC_CBout, sIPC_CB, rIPC_CBin,
IPC_CBin : Pos # PCsCallbacks;
IndicationCB : IndicationMsg;
proc ControlPC(s:PCState) = (
  (s==PC_Off) ->
    ( rICR_PC(powerOn) . ControlPC(StartingUp) +
      rICR_PC(powerOff) . Illegal . delta +
      rICR_PC_Broadcast(restart) . Illegal . delta +
      rICR_PC_Broadcast(shutdown) . Illegal . delta)
  ) +
  (s==Operational) ->
    ( rICR_PC(powerOn) . Illegal . delta +
      rICR_PC(powerOff) . ControlPC(PC_Off) +
      sICR_PC_CB(controlPowerOff) . ControlPC(WaitingShutdown) +
      rICR_PC_Broadcast(restart) . ControlPC(StartingUp) +
      rICR_PC_Broadcast(shutdown) . ControlPC(OS_Shutdown)
    ) +
  (s==WaitingShutdown) ->
    ( rICR_PC(powerOn) . Illegal . delta +
      rICR_PC(powerOff) . ControlPC(PC_Off) +
      rICR_PC_Broadcast(restart) . ControlPC(StartingUp) +
      rICR_PC_Broadcast(shutdown) . ControlPC(OS_Shutdown)
    ) +
  (s==OS_Shutdown) ->
    ( rICR_PC(powerOn) . Illegal . delta +
      rICR_PC(powerOff) . ControlPC(PC_Off) +
      rICR_PC_Broadcast(restart) . Illegal . delta +
      rICR_PC_Broadcast(shutdown) . Illegal . delta
    ) +
  (s==StartingUp) ->
    ( rICR_PC(powerOn) . Illegal . delta +
      rICR_PC(powerOff) . ControlPC(PC_Off) +
      rICR_PC_Broadcast(restart) . Illegal . delta +
      rICR_PC_Broadcast(shutdown) . Illegal . delta +
      sICR_PC_CB(started) . ControlPC(Operational)
    )
) ;

proc GeoPC(s:PCState) = (
  (s==PC_Off) ->
    ( rIGeoPC_Broadcast(shutdown) . Illegal . delta +
      rIGeoPC(powerOn) . GeoPC(StartingUp) +
      rIGeoPC(onPressed) . Illegal . delta +
      rIGeoPC(powerOff) . Illegal . delta +
      rIGeoPC_Broadcast(restart) . Illegal . delta
    ) +
  (s==Operational) ->
    ( rIGeoPC(powerOn) . Illegal . delta +

\begin{verbatim}
80 rIGeoPC(powerOff) . GeoPC(PC_Off) +
81 rIGeoPC(onPressed) . GeoPC(Operational) + % it can happen if the restart is issued before
82 rIGeoPC_Broadcast(shutdown) . GeoPC(OS_Shutdown) +
83 rIGeoPC_Broadcast(restart) . GeoPC(StartingUp) +
84 sIGeoPC_CB(stop) . GeoPC(StopPressed)
85 ) +
86 (s==StopPressed) ->
87 ( rIGeoPC(powerOn) . Illegal . delta +
88 rIGeoPC(powerOff) . GeoPC(PC_Off) +
89 rIGeoPC_Broadcast(shutdown) . GeoPC(OS_Shutdown) +
90 rIGeoPC_Broadcast(restart) . GeoPC(StartingUp) +
91 rIGeoPC(onPressed) . GeoPC(Operational)
92 ) +
93 (s==OS_Shutdown) ->
94 ( rIGeoPC(powerOn) . Illegal . delta +
95 rIGeoPC(powerOff) . GeoPC(PC_Off) +
96 rIGeoPC(onPressed) . Illegal . delta +
97 rIGeoPC_Broadcast(shutdown) . Illegal . delta +
98 rIGeoPC_Broadcast(restart) . Illegal . delta
99 ) +
100 (s==StartingUp) ->
101 ( rIGeoPC(powerOn) . Illegal . delta +
102 rIGeoPC(powerOff) . GeoPC(PC_Off) +
103 rIGeoPC_Broadcast(shutdown) . GeoPC(StartingUp) +
104 rIGeoPC_Broadcast(restart) . Illegal . delta +
105 sIGeoPC_CB(started) . GeoPC(Operational)
106 )
107 );
108
proc NormalPC(id:Pos,s:PCState) =
109 (s==PC_Off) ->
110 ( rIPC(id,powerOn) . NormalPC(id,StartingUp) +
111 rIPC(id,powerOff) . Illegal . delta +
112 rIPC_Broadcast(id,shutdown) . Illegal . delta +
113 rIPC_Broadcast(id,restart) . Illegal . delta
114 ) +
115 (s==Operational) ->
116 ( rIPC(id,powerOn) . Illegal . delta +
117 rIPC(id,powerOff) . NormalPC(id,PC_Off) +
118 rIPC_Broadcast(id,shutdown) . NormalPC(id,OS_Shutdown) +
119 rIPC_Broadcast(id,restart) . NormalPC(id,StartingUp)
120 ) +
121 (s==OS_Shutdown) ->
122 ( rIPC(id,powerOn) . Illegal . delta +
123 rIPC(id,powerOff) . NormalPC(id,PC_Off) +
124 rIPC_Broadcast(id,shutdown) . Illegal . delta +
125 rIPC_Broadcast(id,restart) . Illegal . delta
126 ) +
127 (s==StartingUp) ->
128 ( rIPC(id,powerOn) . Illegal . delta +
129 rIPC(id,powerOff) . NormalPC(id,PC_Off) +
130 rIPC_Broadcast(id,shutdown) . NormalPC(id,StartingUp) +
131 rIPC_Broadcast(id,restart) . Illegal . delta +
132 sIPC_CB(id,started) . NormalPC(id,Operational)
133 )
134 )
135
% the PDU design
136 proc PDU_State_Machine(s:PDUState,geopcOn,crpcOn,geoPressed:Boolean,startedPc:Nat) =
137 (s==PDU_Off) ->
138 ( rIPDU(PDU_switchOn) . sICR_PC(powerOn) . IndicationCB(startingUp) .
139 PDU_State_Machine(StartingUp_CR_PC,geopcOn,crpcOn,geoPressed,startedPc) +
140 rICR_PC_CB(controlPowerOff) . PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) +
141 rICR_PC_Broadcast(id,shutdown) . NormalPC(id,StartingUp) +
142 rICR_PC_Broadcast(id,restart) . Illegal . delta +
143 sIPC_CB(id,started) . NormalPC(id,Operational)
144 )
145 sum id:Pos . rIPC_CB(id,started). PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc)
146 \end{verbatim}
rIPDU(forcedPowerOff) . PDU_State_Machine(Emergency_Off,geopcOn,crpcOn,geoPressed,startedPc) + 

rIPDU(emergencyOff) . PDU_State_Machine(Emergency_Off,geopcOn,crpcOn,geoPressed,startedPc) + 

rICR_PC_CB(controlPowerOff) . PDU_State_Machine(Emergency_Off,geopcOn,crpcOn,geoPressed,startedPc) + 

rIGeoPC_CB(stop) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) + 

rIGeoPC_CB(started) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) + 

sum id:Pos . rIPC_CB(id,started) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) + 

(s==System_Off) -> 

( 
  rIPDU(PDUswitchOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) + 
  rIPDU(powerOn) . sICR_PC(powerOn) . sGeoPC(powerOn) | scommandhandler(powerOn) . 
  IndicationCB(startingUp) . PDU_State_Machine(StartingUpAllPcs,false,false,false,0) + 
  rIPDU(powerOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) + 
  rIPDU(forcedPowerOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) + 
  rIPDU(emergencyOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) + 
  rICR_PC_CB(controlPowerOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) + 
  rIGeoPC_CB(started) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) + 
  rICR_PC_CB(started) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) + 
  rIGeoPC_CB(started) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) + 
  sum id:Pos . rIPC_CB(id,started) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) + 
)

(s==Geo_Stop) -> 

( 
  rIPDU(PDUswitchOff) . sICR_PC(powerOff) . sGeoPC(powerOff) | scommandhandler(powerOff) . 
  PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) + 
  rIPDU(powerOn) . sICR_PC(onPressed) . IndicationCB(systemOn) . 
  PDU_State_Machine(System_On,geopcOn,crpcOn,false,startedPc) + 
  rIPDU(powerOff) . sICR_PC_Broadcast(shutdown) . sMsghandler(shutdown) . release . 
  sGeoPC(powerOff) | scommandhandler(powerOff) . IndicationCB(systemStandby) . 
  PDU_State_Machine(SystemStandby,geopcOn,crpcOn,geoPressed,startedPc) + 
  rIPDU(powerOff) . sICR_PC(powerOff) . sGeoPC(powerOff) | scommandhandler(powerOff) . 
  IndicationCB(off) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) + 
  rIPDU(powerOff) . sICR_PC(powerOff) . sGeoPC(powerOff) | scommandhandler(powerOff) . 
  IndicationCB(off) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) + 
  rICR_PC_CB(controlPowerOff) . sICR_PC_Broadcast(shutdown) . sMsghandler(shutdown) . release . 
  sICR_PC(powerOff) . sGeoPC(powerOff) | scommandhandler(powerOff) . IndicationCB(off) . 
  PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) + 
  rIGeoPC_CB(started) . Illegal . delta + 
  rIGeoPC_CB(started) . Illegal . delta + 
  sum id:Pos . rIPC_CB(id,started) . Illegal . delta
)

(s==StartingUpAllPcs) -> 

( 
  rIPDU(PDUswitchOff) . sICR_PC(powerOff) . sGeoPC(powerOff) | scommandhandler(powerOff) . 
  PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) + 
  rIPDU(powerOn) . sICR_PC(onPressed) . IndicationCB(systemOn) . 
  PDU_State_Machine(System_On,geopcOn,crpcOn,false,startedPc) + 
  rIPDU(powerOff) . sICR_PC_Broadcast(shutdown) . sMsghandler(shutdown) . release . 
  sGeoPC(powerOff) | scommandhandler(powerOff) . IndicationCB(systemStandby) . 
  PDU_State_Machine(SystemStandby,geopcOn,crpcOn,geoPressed,startedPc) + 
  rIPDU(powerOff) . sICR_PC(powerOff) . sGeoPC(powerOff) | scommandhandler(powerOff) . 
  IndicationCB(systemStandby) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) + 
  (! crpcOn ) -> rIPDU(powerOff) . sGeoPC_Broadcast(shutdown) . sMsghandler(shutdown) . release . 
  sGeoPC(powerOff) | scommandhandler(powerOff) . IndicationCB(shutdown) . 
  PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) + 
  (! crpcOn ) -> rIPDU(powerOff) . sGeoPC_Broadcast(shutdown) . sMsghandler(shutdown) . release . 
  sGeoPC(powerOff) | scommandhandler(powerOff) . IndicationCB(shutdown) . 
  PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) + 
  rIPDU(forcedPowerOff) . sICR_PC(powerOff) . sGeoPC(powerOff) | scommandhandler(powerOff) . 
  IndicationCB(off) . 
  PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) + 
  rIPDU(emergencyOff) . sICR_PC(powerOff) . sGeoPC(powerOff) | scommandhandler(powerOff) . 
  IndicationCB(off) . 
  PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) + 
  rICR_PC_CB(controlPowerOff). sICR_PC_Broadcast(shutdown) . sMsghandler(shutdown) . release . 
  sICR_PC(powerOff) . sGeoPC_Broadcast(shutdown) . sMsghandler(shutdown) . release . 
)
[]
D  Synchronous push model

sort Command = struct PDUswitchOn | PDUswitchOff | powerOn | powerOff | forcedPowerOff | emergencyOff | onPressed;
Bmsg = struct restart | shutdown ;
PCsCallbacks = struct controlPowerOff | stop | started ;
PCState = struct PC_Off | Operational | WaitingShutdown | StartingUp | OS_Shutdown | StopPressed ;
IndicationMsg = struct startingUp | off | systemOn | systemStandby | geoStop ;

act Illegal,
srelease,rrelease,release;
IPDU,
sICR_PC,rICR_PC,ICR_PC,
sGeoPC,rGeoPC,IGeoPC,
commandhandler, rcommandhandler, commandhandler : Command;

PROCControlPC(s:PCState) = (s==PC_Off) ->
( rICR_PC(powerOn) . ControlPC(StartingUp) +
 rICR_PC(powerOff) . Illegal . delta +
 rICR_PC_Broadcast(restart) . Illegal . delta +
 rICR_PC_Broadcast(shutdown) . Illegal . delta ) +
(s==Operational) ->
( rICR_PC(powerOn) . Illegal . delta +
rICR_PC(powerOff) . ControlPC(PC_Off) +
sICR_PC_CB(controlPowerOff) . ControlPC(WaitingShutdown) +
rICR_PC_Broadcast(restart) . ControlPC(StartingUp) +
rICR_PC_Broadcast(shutdown) . ControlPC(OS_Shutdown)
)
)

(s==WaitingShutdown) ->
( rICR_PC(powerOn) . Illegal . delta +
 rICR_PC(powerOff) . ControlPC(PC_Off) +
rICR_PC_Broadcast(restart) . ControlPC(StartingUp) +
rICR_PC_Broadcast(shutdown) . ControlPC(OS_Shutdown)
)

(s==OS_Shutdown) -> (sICR_PC_CB(started) . ControlPC(Operational)

)

)

proc GeoPC(s:PCState) =

(a==PC_Off) ->
( rIGeoPC_Broadcast(shutdown) . Illegal . delta +
 rIGeoPC(powerOn) . GeoPC(StartingUp) +
rIGeoPC(onPressed) . Illegal . delta +
 rIGeoPC(powerOff) . Illegal . delta +
rIGeoPC_Broadcast(restart) . Illegal . delta +
)

(a==Operational) ->
( rIGeoPC(powerOn) . Illegal . delta +
 rIGeoPC(powerOff) . GeoPC(PC_Off) +
rIGeoPC(onPressed) . GeoPC(Operational) + % it can happen if the restart is issued before
 rIGeoPC_Broadcast(shutdown) . GeoPC(OS_Shutdown) +
rIGeoPC_Broadcast(restart) . GeoPC(StartingUp) +
 sIGeoPC_CB(stop) . GeoPC(StopPressed)
)

(a==StopPressed) ->
( rIGeoPC(powerOn) . Illegal . delta +
 rIGeoPC(powerOff) . GeoPC(PC_Off) +
rIGeoPC_Broadcast(shutdown) . GeoPC(OS_Shutdown) +
rIGeoPC_Broadcast(restart) . GeoPC(StartingUp) +
rIGeoPC(onPressed) . GeoPC(Operational)
)

(a==OS_Shutdown) ->
( rIGeoPC(powerOn) . Illegal . delta +
 rIGeoPC(powerOff) . GeoPC(PC_Off) +
rIGeoPC(onPressed) . Illegal . delta +
rIGeoPC_Broadcast(shutdown) . Illegal . delta +
rIGeoPC_Broadcast(restart) . Illegal . delta
)

(a==StartingUp) ->
( rIGeoPC(powerOn) . Illegal . delta +
 rIGeoPC(powerOff) . GeoPC(PC_Off) +
rIGeoPC_Broadcast(shutdown) . GeoPC(OS_Shutdown) +
rIGeoPC_Broadcast(restart) . Illegal . delta +
sIGeoPC_CB(started) . GeoPC(Operational)
)

);
(s==PC_Off) ->
  ( rIPC(id,powerOn) . NormalPC(id,StartingUp) +
    rIPC(id,powerOff) . Illegal . delta +
    rIPC_Broadcast(id,shutdown) . Illegal . delta +
    rIPC_Broadcast(id,restart) . Illegal . delta ) +
  (s==Operational) ->
  ( rIPC(id,powerOn) . Illegal . delta +
    rIPC(id,powerOff) . NormalPC(id,PC_Off) +
    rIPC_Broadcast(id,shutdown) . NormalPC(id,OS_Shutdown) +
    rIPC_Broadcast(id,restart) . Illegal . delta +
    sIPC_CB(id,started) . NormalPC(id,Operational) ) +
  (s==OS_Shutdown) ->
  ( rIPC(id,powerOn) . Illegal . delta +
    rIPC(id,powerOff) . NormalPC(id,PC_Off) +
    rIPC_Broadcast(id,shutdown) . Illegal . delta +
    rIPC_Broadcast(id,restart) . Illegal . delta ) +
  (s==StartingUp) ->
  ( rIPC(id,powerOn) . Illegal . delta +
    rIPC(id,powerOff) . NormalPC(id,PC_Off) +
    rIPC_Broadcast(id,shutdown) . NormalPC(id,StartingUp) +
    rIPC_Broadcast(id,restart) . Illegal . delta +
    sIPC_CB(id,started) . NormalPC(id,Operational) ) +
%
% the PDU design
proc PDU_State_Machine(s:PDUState,geopcOn,crpcOn,geoPressed:Bool,startedPc:Nat) =
  (s==PDU_Off) ->
    ( IPDU(PDUswitchOn) . sICR_PC(powerOn) . IndicationCB(startingUp) .
      PDU_State_Machine(StartingUp_CR_PC,geopcOn,crpcOn,geoPressed,startedPc) +
      rICR_PC_CB(controlPowerOff) . PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) +
      rGeoPC_CB(stop) . PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) +
      rGeoPC_CB(started) . PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) +
      sum id:Pos . rIPC_CB(id,started) .
      PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) ) +
  (s==StartingUp_CR_PC) ->
    IPDU(PDUswitchOff) . sICR_PC(powerOff) .
    PDU_State_Machine(StartingUp_CR_PC,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(powerOn) . PDU_State_Machine(StartingUp_CR_PC,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(powerOff) . PDU_State_Machine(StartingUp_CR_PC,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(forcedPowerOff) . sICR_PC(powerOff) . IndicationCB(off) .
    PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(emergencyOff) . sICR_PC(powerOff) . IndicationCB(off) .
    PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    rICR_PC_CB(controlPowerOff) . Illegal . delta +
    rICR_PC_CB(started) . IndicationCB(systemStandby) .
    PDU_State_Machine(SystemStandby,geopcOn,crpcOn,geoPressed,startedPc) +
    rGeoPC_CB(stop) . PDU_State_Machine(StartingUp_CR_PC,geopcOn,crpcOn,geoPressed,startedPc) +
    rGeoPC_CB(started) . PDU_State_Machine(StartingUp_CR_PC,geopcOn,crpcOn,geoPressed,startedPc) +
    sum id:Pos . rIPC_CB(id,started) .
    PDU_State_Machine(StartingUp_CR_PC,geopcOn,crpcOn,geoPressed,startedPc) +
  (s==SystemStandby) ->
    IPDU(PDUswitchOff) . sICR_PC(powerOff) .
    PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(powerOn) . sIGeoPC(powerOn) . scommandhandler(powerOn) . release . IndicationCB(startingUp) .
    PDU_State_Machine(StartingUpAllPcs,false,true,false,0) +
    IPDU(powerOff) . PDU_State_Machine(SystemStandby,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(forcedPowerOff) . sICR_PC(powerOff) . IndicationCB(off) .
    PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(emergencyOff) . sICR_PC(powerOff) . IndicationCB(off) .

PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    rICR_PC_CB(controlPowerOff) . sICR_PC_Broadcast(shutdown) . sICR_PC(powerOff).
    IndicationCB(off) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    rICR_PC_CB(stop) . PDU_State_Machine(SystemStandby,geopcOn,crpcOn,geoPressed,startedPc) +
    sICR_PC_Broadcast(shutdown) . sICR_PC(powerOff) .
    rRelease . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    sum id:Pos . rICR_PC_CB(id,started).
    PDU_State_Machine(SystemStandby,geopcOn,crpcOn,geoPressed,startedPc) +

    )
    (s==System_On) ->
    (IPDU(PDUswitchOff) . sICR_PC(powerOff) . sGeoPC(powerOff) . sCommandHandler(powerOff) .
        rRelease . PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(powerOn) . sICR_PC_Broadcast(restart) . sGeoPC_Broadcast(restart) .
        sMshandler(restart) . sMshandler(shutdown) . release .
    sGeoPC(powerOff) . sCommandHandler(powerOff) . release . IndicationCB(systemStandby) .
    PDU_State_Machine(SystemStandby,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(forcedPowerOff) . sICR_PC_Broadcast(shutdown) . sGeoPC_Broadcast(shutdown) .
        sMshandler(shutdown) . release .
    rICR_PC_CB(started) . Illegal . delta +
    rIGeoPC_CB(started) . Illegal . delta +
    sum id:Pos . rIPC_CB(id,started) . Illegal . delta

    )
    (s==Emergency_Off) ->
    (IPDU(PDUswitchOff) . PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(powerOn) . sICR_PC(powerOn) . sGeoPC(powerOn) . sCommandHandler(powerOn) .
        rRelease . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(emergencyOff) . sICR_PC(powerOff) .
        sCommandHandler(powerOff) . release . IndicationCB(systemStandby) .
    PDU_State_Machine(SystemStandby,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(emergencyOff) . sICR_PC_Broadcast(shutdown) . sGeoPC_Broadcast(shutdown) .
        sMshandler(shutdown) . release . sICR_PC(powerOff) . sGeoPC(powerOff) .
        rRelease . IndicationCB(off) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(forcedPowerOff) . sICR_PC_Broadcast(shutdown) . sGeoPC_Broadcast(shutdown) .
        sMshandler(shutdown) . release .
    rICR_PC_CB(controlPowerOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    rICR_PC_CB(stop) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    rIGeoPC_CB(started) . IndicationCB(geoStop) .
    PDU_State_Machine(Geo_Stop,geopcOn,crpcOn,geoPressed,startedPc) +
    rICR_PC_CB(started) . Illegal . delta +
    rIGeoPC_CB(started) . Illegal . delta +
    sum id:Pos . rIPC_CB(id,started) . Illegal . delta

    )
    (s==System_Off) ->
    (IPDU(PDUswitchOff) . PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(powerOn) . sICR_PC(powerOn) . sGeoPC(powerOn) .
        sCommandHandler(powerOn) . rRelease . PDU_State_Machine(Emergency_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(emergencyOff) . sICR_PC(powerOff) .
        sCommandHandler(powerOff) . rRelease . IndicationCB(systemStandby) .
    PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(forcedPowerOff) . sICR_PC_Broadcast(shutdown) . sGeoPC_Broadcast(shutdown) .
        sMshandler(shutdown) . release . sICR_PC(powerOff) . sGeoPC(powerOff) .
        rRelease . IndicationCB(off) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(emergencyOff) . sICR_PC_Broadcast(shutdown) . sGeoPC_Broadcast(shutdown) .
        sMshandler(shutdown) . release .
    rICR_PC_CB(controlPowerOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    rICR_PC_CB(stop) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    rIGeoPC_CB(started) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    rIGeoPC_CB(started) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    sum id:Pos . rIPC_CB(id,started) .
    PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +

    )
    (s==Geo_Stop) ->
    (IPDU(PDUswitchOff) . sICR_PC(powerOff) . sGeoPC(powerOff) .
        sCommandHandler(powerOff) .
        rRelease . PDU_State_Machine(Emergency_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(powerOn) . sICR_PC(powerOn) . sGeoPC(powerOn) .
        sCommandHandler(powerOn) . rRelease . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(emergencyOff) . sICR_PC(powerOff) .
        sCommandHandler(powerOff) . rRelease . IndicationCB(systemStandby) .
    PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(forcedPowerOff) . sICR_PC_Broadcast(shutdown) . sGeoPC_Broadcast(shutdown) .
        sMshandler(shutdown) . release . sICR_PC(powerOff) . sGeoPC(powerOff) .
        rRelease . IndicationCB(off) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(emergencyOff) . sICR_PC_Broadcast(shutdown) . sGeoPC_Broadcast(shutdown) .
        sMshandler(shutdown) . release .
    rICR_PC_CB(controlPowerOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    rICR_PC_CB(stop) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    rIGeoPC_CB(started) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    rIGeoPC_CB(started) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    sum id:Pos . rIPC_CB(id,started) .
    PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +

    )
    (s==Emergency_Off) ->
    (IPDU(PDUswitchOff) . PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(powerOn) . sICR_PC(powerOn) . sGeoPC(powerOn) .
        sCommandHandler(powerOn) . rRelease . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(emergencyOff) . sICR_PC(powerOff) .
        sCommandHandler(powerOff) . rRelease . IndicationCB(systemStandby) .
    PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(forcedPowerOff) . sICR_PC_Broadcast(shutdown) . sGeoPC_Broadcast(shutdown) .
        sMshandler(shutdown) . release . sICR_PC(powerOff) . sGeoPC(powerOff) .
        rRelease . IndicationCB(off) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    IPDU(emergencyOff) . sICR_PC_Broadcast(shutdown) . sGeoPC_Broadcast(shutdown) .
        sMshandler(shutdown) . release .
    rICR_PC_CB(controlPowerOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    rICR_PC_CB(stop) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    rIGeoPC_CB(started) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    rIGeoPC_CB(started) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    sum id:Pos . rIPC_CB(id,started) .
    PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +

    )
    (s==Geo_Stop) ->
    (IPDU(PDUswitchOff) . sICR_PC(powerOff) . sGeoPC(powerOff) .
        sCommandHandler(powerOff) .
        rRelease . PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) +
IPDU(powerOn) . sIGeoPC(onPressed) . IndicationCB(systemOn) .

PDU_State_Machine(System_On,geopcOn,crpcOn,false,startedPc) +
IPDU(powerOff) . sIGeoPC_Broadcast(shutdown) . sMshandler(shutdown) . release .
RIndicationCB(powerOff) . scommandhandler(powerOff) . release .

PDU_State_Machine(SystemStandby,geopcOn,crpcOn,geoPressed,startedPc) +
IPDU(forcedPowerOff) . sICR_PC(powerOff) . sIGeoPC(powerOff) . scommandhandler(powerOff) .
release . IndicationCB(off) .

PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
IPDU(emergencyOff) . sICR_PC(powerOff) . sIGeoPC(powerOff) . scommandhandler(powerOff) .
release . IndicationCB(off) .

PDU_State_Machine(Emergency_Off,geopcOn,crpcOn,geoPressed,startedPc) +
rICR_PC_CB(controlPowerOff) . sICR_PC_Broadcast(shutdown) . sIGeoPC_Broadcast(shutdown) .
sMshandler(shutdown) . release . sICR_PC(powerOff) . sIGeoPC(powerOff) .
scommandhandler(powerOff) . release . IndicationCB(off) .

PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
IPDU(emergencyOff) . sICR_PC(powerOff) . sIGeoPC(powerOff) . scommandhandler(powerOff) .
release . IndicationCB(off) .

PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
IPDU(forcedPowerOff) . sICR_PC(powerOff) . sIGeoPC(powerOff) . scommandhandler(powerOff) .
release . IndicationCB(off) .

PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
IPDU(powerOff) . sICR_PC(powerOff) . sIGeoPC(powerOff) . scommandhandler(powerOff) .
release . IndicationCB(systemStandby) .

PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
IPDU(SwitchOff) . sICR_PC(powerOff) . sIGeoPC(powerOff) . scommandhandler(powerOff) .
release . PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) +

PDU_State_Machine(startingUpAllPcs,geopcOn,crpcOn,geoPressed,startedPc) +
IPDU(powerOn) . PDU_State_Machine(startingUpAllPcs,geopcOn,crpcOn,geoPressed,startedPc) +

(startedPc==5 && startedPc<5) -> rICR_PC_CB(started) . IndicationCB(geoStop) .
PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +

(startedPc<4) -> sum id:Pos . rIPC_CB(id,started) .
PDU_State_Machine(startingUpAllPcs,geopcOn,crpcOn,geoPressed,startedPc) +

(startedPc==4 && ! geoPressed) -> sum id:Pos . rIPC_CB(id,started) .
PDU_State_Machine(startingUpAllPcs,geopcOn,crpcOn,geoPressed,startedPc) +

((startedPc==4 && geoPressed) || ! crpcOn) -> rIGeoPC_CB(started) . IndicationCB(systemOn) .
PDU_State_Machine(startingUpAllPcs,geopcOn,crpcOn,geoPressed,startedPc) +

(startedPc<4) -> sum id:Pos . rIPC_CB(id,started) .
PDU_State_Machine(startingUpAllPcs,geopcOn,crpcOn,geoPressed,startedPc) +

(startedPc==4 && ! geoPressed) -> sum id:Pos . rIPC_CB(id,started) .
PDU_State_Machine(startingUpAllPcs,geopcOn,crpcOn,geoPressed,startedPc) +

(startedPc==4 && geoPressed) -> sum id:Pos . rIPC_CB(id,started) .
PDU_State_Machine(startingUpAllPcs,geopcOn,crpcOn,geoPressed,startedPc) +

((startedPc==4 && geoPressed) || ! crpcOn) -> sum id:Pos . rIPC_CB(id,started) .
PDU_State_Machine(startingUpAllPcs,geopcOn,crpcOn,geoPressed,startedPc) +
proc Handler = sum c:Command . rcommandhandler(c) | sIPC(1,c) . sIPC(2,c) . sIPC(3,c) .
                     sIPC(4,c) . srelease|sIPC(5,c) . Handler +
sum m:Bmsg . rMsghandler(m) | sIPC_Broadcast(1,m) . sIPC_Broadcast(2,m) .
                     sIPC_Broadcast(3,m) . sIPC_Broadcast(4,m) . srelease|sIPC_Broadcast(5,m) . Handler;

proc System=
   hide({ ICR_PC,ICR_PC_Broadcast,ICR_PC_CB,
          IGeoPC,IGeoPC_Broadcast,IGeoPC_CB,
          IPC,IPC_Broadcast,IPC_CB,
          release,Msghandler,commandhandler
       },
   allow({IPDU,
          ICR_PC,ICR_PC_Broadcast,ICR_PC_CB,
          IGeoPC,IGeoPC_Broadcast,IGeoPC_CB,
          IPC,IPC_Broadcast,IPC_CB,
          IndicationCB,
          commandhandler|IPC,
          Msghandler|IPC_Broadcast,
          release|IPC,release|IPC_Broadcast,
          Illegal
       },
   comm({sICR_PC|rICR_PC->ICR_PC,
          sICR_PC_Broadcast|rICR_PC_Broadcast->ICR_PC_Broadcast,
          sIGeoPC|rICR_PC_Broadcast->ICR_PC_Broadcast,
          sICR_PC_CB|rICR_PC_CB->ICR_PC_CB,
          sICR_PC_CB|rICR_PC_CB->ICR_PC_CB,
          sIPC_CB | rIPC_CB -> IPC_CB,
          srelease | rrelease ->release,
          rIPC|sIPC->IPC,
          rIPC_Broadcast|sIPC_Broadcast->IPC_Broadcast,
          sMsghandler | rMsghandler -> Msghandler,
          scommandhandler | rcommandhandler -> commandhandler
       },
   | ControlPC(PC_Off) ||
   | GeoPC(PC_Off) ||
   | NormalPC(1,PC_Off) ||
   | NormalPC(2,PC_Off) ||
   | NormalPC(3,PC_Off) ||
   | NormalPC(4,PC_Off) ||
   | NormalPC(5,PC_Off) ||
   | Handler ||
   | PDU_State_Machine(PDU_Off,false,false,false,0)))));

init System;

E  Synchronous push model with global synchronous communication

sort Command = struct PDUswitchOn | PDUswitchOff | powerOn | powerOff | forcedPowerOff |
                    emergencyOff | onPressed;
Bmsg = struct restart | shutdown ;
PCsCallbacks = struct controlPowerOff | stop | started ;
PCState = struct PC_Off | Operational | WaitingShutdown | StartingUp | OS_Shutdown | StopPressed;
PDUState = struct PDU_Off | StartingUp_CR_PC | StartingUpAllPcs | SystemStandby | System_On |
        Emergency_Off | System_Off | Geo_Stop ;
IndicationMsg = struct startingUp | off | systemOn | systemStandby | geoStop ;
source, release, release;
IPDU,
sICR_PC, rICR_PC, ICR_PC,
sGeoPC, rGeoPC, GeoPC,
commandhandler, rcommandhandler, commandhandler : Command;
sIPC, rIPC, IPC : Pos # Command;
sICR_PC_Broadcast, rICR_PC_Broadcast, ICR_PC_Broadcast,
sGeoPC_Broadcast, rGeoPC_Broadcast, GeoPC_Broadcast,
messagehandler, Msghandler, Msghandler : Event;
sIPC_Broadcast, rIPC_Broadcast, IPC_Broadcast : Pos # Event;
sICR_PC_CB, rICR_PC_CB, ICR_PC_CB,
sGeoPC_CB, rGeoPC_CB, GeoPC_CB : PCsCallbacks;
sIPC_CB, rIPC_CB, IPC_CB : Pos # PCsCallbacks;
IndicationCB : IndicationMsg;

proc ControlPC(s:PCState) =

( s==PC_Off ) ->
  ( rICR_PC(powerOn). ControlPC(StartingUp) +
    rICR_PC(powerOff). Illegal . delta +
    rICR_PC_Broadcast(restart). Illegal . delta +
    rICR_PC_Broadcast(shutdown). Illegal . delta ) +

( s==Operational ) ->
  ( rICR_PC(powerOn). Illegal . delta +
    rICR_PC(powerOff). ControlPC(PC_Off) +
    rICR_PC_CB(controlPowerOff). ControlPC(WaitingShutdown) +
    rICR_PC_Broadcast(restart). ControlPC(StartingUp) +
    rICR_PC_Broadcast(shutdown). ControlPC(OS_Shutdown) ) +

( s==WaitingShutdown ) ->
  ( rICR_PC(powerOn). Illegal . delta +
    rICR_PC(powerOff). ControlPC(PC_Off) +
    rICR_PC_Broadcast(restart). Illegal . delta +
    rICR_PC_Broadcast(shutdown). Illegal . delta ) +

( s==OS_Shutdown ) ->
  ( rICR_PC(powerOn). Illegal . delta +
    rICR_PC(powerOff). ControlPC(PC_Off) +
    rICR_PC_Broadcast(restart). Illegal . delta +
    rICR_PC_Broadcast(shutdown). Illegal . delta +
    sICR_PC_CB(started). ControlPC(Operational) ) +

( s==StartingUp ) ->
  ( rICR_PC(powerOn). Illegal . delta +
    rICR_PC(powerOff). ControlPC(PC_Off) +
    rICR_PC_Broadcast(restart). Illegal . delta +
    rICR_PC_Broadcast(shutdown). Illegal . delta +
    sICR_PC_CB(started). ControlPC(Operational) ) +

);
(s==StopPressed) ->
  ( rIGeoPC(powerOn). Illegal. delta +
  rIGeoPC(powerOff). GeoPC(PC_Off) +
  rIGeoPC_Broadcast(shutdown). GeoPC(OS_Shutdown) +
  rIGeoPC_Broadcast(restart). GeoPC(StartingUp) +
  rIGeoPC(onPressed). GeoPC(Operational)
)+
(s==OS_Shutdown) ->
  ( rIGeoPC(powerOn). Illegal. delta +
  rIGeoPC(powerOff). GeoPC(PC_Off) +
  rIGeoPC(onPressed). Illegal. delta +
  rIGeoPC_Broadcast(shutdown). Illegal. delta +
  rIGeoPC_Broadcast(restart). Illegal. delta
  )+
(s==StartingUp) ->
  ( rIGeoPC(powerOn). Illegal. delta +
  rIGeoPC(powerOff). GeoPC(PC_Off) +
  rIGeoPC_Broadcast(shutdown). GeoPC(StartingUp) +
  rIGeoPC_Broadcast(restart). Illegal. delta +
  sIGeoPC_CB(started). GeoPC(Operational)
  )
)

proc NormalPC(id:Pos,s:PCState) =
  (s==PC_Off) ->
    ( rIPC(id,powerOn). NormalPC(id,StartingUp) +
    rIPC(id,powerOff). Illegal. delta +
    rIPC_Broadcast(id,shutdown). Illegal. delta +
    rIPC_Broadcast(id,restart). Illegal. delta
    )+
  (s==Operational) ->
    ( rIPC(id,powerOn). Illegal. delta +
    rIPC(id,powerOff). NormalPC(id,PC_Off) +
    rIPC_Broadcast(id,shutdown). NormalPC(id,OS_Shutdown) +
    rIPC_Broadcast(id,restart). NormalPC(id,StartingUp)
    )+
  (s==OS_Shutdown) ->
    ( rIPC(id,powerOn). Illegal. delta +
    rIPC(id,powerOff). NormalPC(id,PC_Off) +
    rIPC_Broadcast(id,shutdown). Illegal. delta +
    rIPC_Broadcast(id,restart). Illegal. delta
    )+
  (s==StartingUp) ->
    ( rIPC(id,powerOn). Illegal. delta +
    rIPC(id,powerOff). NormalPC(id,PC_Off) +
    rIPC_Broadcast(id,shutdown). NormalPC(id,StartingUp) +
    rIPC_Broadcast(id,restart). Illegal. delta +
    sIPC_CB(id,started). NormalPC(id,Operational)
    )
  )

% the PDU design

proc PDU_State_Machine(s:PDUState,geopcOn,crpcOn,geoPressed:Bool,startedPc:Nat) =
  (s==PDU_Off) ->
    ( rPDU(PDUswitchOn) . sICR_PC(powerOn) . IndicationCB(startingUp) .
    PDU_State_Machine(StartingUp_CR_PC,geopcOn,crpcOn,geoPressed,startedPc) +
    rICR_PC_CB(controlPowerOff) .
    PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    rIGeoPC_CB(stop) . PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    rICR_PC_CB(started) . PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) +
    sum id:Pos . rIPC_CB(id,started) .
    PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc)
  )+
  (s==StartingUp_CR_PC) ->
    ( rPDU(PDUswitchOff) . sICR_PC(powerOff) .
  )
PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) +
IPDU(powerOn) . PDU_State_Machine(StartingUp_CR_PC,geopcOn,crpcOn,geoPressed,startedPc) +
IPDU(powerOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
IPDU(forcedPowerOff) . rICR_PC_Broadcast(systemStandby) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
IPDU(emergencyOff) . rICR_PC_Broadcast(shutdown) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
IPDU(PDUswitchOff) . rICR_PC_CB(controlPowerOff) . Illegal . delta +
( s==SystemStandby ) ->
( IPDU(PDUswitchOff) . sICR_PC(powerOff) . PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) +
IPDU(powerOn) . sICR_PC(powerOn) | scommandhandler(powerOn) . PDU_State_Machine(StartingUpAllPcs,false,true,true,0) +
IPDU(powerOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
IPDU(forcedPowerOff) . rICR_PC_CB(controlPowerOff) . Illegal . delta +
( s==System_On ) ->
( IPDU(PDUswitchOff) . sICR_PC(powerOff) . sICR_PC_Broadcast(restart) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +
IPDU(powerOn) . sICR_PC_Broadcast(shutdown) . sICR_PC(powerOff) . rICR_PC_CB(started) . Illegal . delta +
( a==System_Standby ) ->
PDU_State_Machine(Emergency_Off,geopcOn,crpcOn,geoPressed,startedPc) +

rIGeoPC_CB(stop) . PDU_State_Machine(Emergency_Off,geopcOn,crpcOn,geoPressed,startedPc) +

rICR_PC_CB(started) . PDU_State_Machine(Emergency_Off,geopcOn,crpcOn,geoPressed,startedPc) +

rIGeoPC_CB(started) . PDU_State_Machine(Emergency_Off,geopcOn,crpcOn,geoPressed,startedPc) +

sum id:Pos . rIPC_CB(id,started) .

PDU_State_Machine(Emergency_Off,geopcOn,crpcOn,geoPressed,startedPc)

) +

(s==System_Off) ->

( IPDU(PDUswitchOff) . PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) +

IPDU(powerOn) . sICR_PC(powerOn) . sIGeoPC(powerOn) | sc mandhandler(powerOn) .

IndicationCB(startingUp) . PDU_State_Machine(StartingUpAllPcs,false,false,false,0) +

IPDU(powerOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +

IPDU(forcedPowerOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +

IPDU(emergencyOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +

rICR_PC_CB(controlPowerOff) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +

rIGeoPC_CB(started) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +

sum id:Pos . rIPC_CB(id,started) .

PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc)

) +

(s==Geo_Stop) ->

( IPDU(PDUswitchOff) . sICR_PC(powerOff) . sIGeoPC(powerOff) | sc mandhandler(powerOff) .

IPDU(powerOn) . sIGeoPC(onPressed) . IndicationCB(systemOn) .

IPDU(powerOff) . sIGeoPC_Broadcast(shutdown) . sMsghandler(shutdown) . rrelease .

sIGeoPC(powerOff) | sc mandhandler(powerOff) . IndicationCB(systemStandby).

IPDU(forcedPowerOff) . sICR_PC(powerOff) . sIGeoPC(powerOff) | sc mandhandler(powerOff) .

IndicationCB(off) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +

IPDU(emergencyOff) . sICR_PC(powerOff) . sIGeoPC(powerOff) | sc mandhandler(powerOff) .

IndicationCB(off) . PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc) +

IPDU(powerOff) . sIGeoPC_Broadcast(shutdown) . sMsghandler(shutdown) . rrelease .

sICR_PC(powerOff) . sIGeoPC(powerOff) | sc mandhandler(powerOff) . IndicationCB(off).

PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc)

) +

(s==StartingUpAllPcs) ->

( IPDU(PDUswitchOff) . sICR_PC(powerOff) . sIGeoPC(powerOff) | sc mandhandler(powerOff) .

PDU_State_Machine(PDU_Off,geopcOn,crpcOn,geoPressed,startedPc) +

IPDU(powerOn) . PDU_State_Machine(StartingUpAllPcs,geopcOn,crpcOn,geoPressed,startedPc) +

(crpcOn ) -> IPDU(powerOff) . sIGeoPC_Broadcast(shutdown) . sMsghandler(shutdown) .

-release . sIGeoPC(powerOff) | sc mandhandler(powerOff) . IndicationCB(systemStandby).

IPDU(powerOff) . sIGeoPC_Broadcast(shutdown) . sMsghandler(shutdown) .

((geopcOn && startedPc==5 && geoPressed)) -> rICR_PC_CB(started) . IndicationCB(geoStop).

PDU_State_Machine(System_Off,geopcOn,crpcOn,geoPressed,startedPc)

) +

((geopcOn && startedPc==5 && geoPressed)) -> rICR_PC_CB(started). IndicationCB(geoStop).

PDU_State_Machine(geoStop,geopcOn,crpcOn,geoPressed,startedPc) +
((geopcOn && startedPc<5 && ! geoPressed)) -> rICR_PC_CB(started) .
IndicationCB(systemOn) .
PDU_State_Machine(System_On,geopcOn,crpcOn,geoPressed,startedPc) +
((! geopcOn || startedPc<5)) -> rICR_PC_CB(started) .
PDU_State_Machine(StartingUpAllPcs,geopcOn,crpcOn,geoPressed,startedPc) +
(rIGeoPC_CB(stop) . PDU_State_Machine(System_On,geopcOn,crpcOn,geoPressed,startedPc) +
((! geopcOn || startedPc<5)) -> rICR_PC_CB(started) .
PDU_State_Machine(StartingUpAllPcs,geopcOn,crpcOn,geoPressed,startedPc) +
(rIGeoPC_CB(stop) . PDU_State_Machine(System_On,geopcOn,crpcOn,geoPressed,startedPc) +
((startedPc<4) -> sum id:Pos . rIPC_CB(id,started) .
PDU_State_Machine(StartingUpAllPcs,geopcOn,crpcOn,geoPressed,startedPc) +
(startedPc==4 && geopcOn && crpcOn && ! geoPressed) -> sum id:Pos . rIPC_CB(id,started) .
IndicationCB(systemOn) .
PDU_State_Machine(System_On,geopcOn,crpcOn,geoPressed,startedPc) +
((startedPc==4 && geopcOn && crpcOn && ! geoPressed) -> sum id:Pos . rIPC_CB(id,started) .
IndicationCB(systemOn) .
PDU_State_Machine(System_On,geopcOn,crpcOn,geoPressed,startedPc) +
((startedPc==4) && (geopcOn==false || ! crpcOn)) -> sum id:Pos . rIPC_CB(id,started) .
PDU_State_Machine(StartingUpAllPcs,geopcOn,crpcOn,geoPressed,startedPc) +
)
;
)

proc Handler = sum c:Command . rcommandhandler(c) | sIPC(1,c) | sIPC(2,c) | sIPC(3,c) | sIPC(4,c) | sIPC(5,c) . Handler +
sim m:Bmsg . rMsghandler(m) | sIPCBroadcast(1,m) . sIPCBroadcast(2,m) .
sIPCBroadcast(3,m) . sIPCBroadcast(4,m) . sIPCBroadcast(5,m) . Handler;

proc System=
hide({ ICR_PC,ICR_PC_Broadcast,ICR_PC_CB,
IGeoPC,IGeoPC_Broadcast,IGeoPC_CB,
IPC,IPC_Broadcast,IPC_CB,
release,Msghandler,commandhandler },
allow({IPDU,
ICR_PC,ICR_PC_Broadcast,ICR_PC_CB,
IGeoPC,IGeoPC_Broadcast,IGeoPC_CB,
IPC,IPC_Broadcast,IPC_CB,
IndicationCB,
IGeoPC|commandhandler|IPC|IPC|IPC|IPC|IPC,
Maghandler|IPC_Broadcast,
release|IPC_Broadcast,
Illegal },
comm({sICR_PC|rICR_PC->ICR_PC,
sICR_PC_Broadcast|rICR_PC_Broadcast->ICR_PC_Broadcast,
sICR_PC_Broadcast|rICR_PC_Broadcast->ICR_PC_Broadcast,
sICR_PC_CB|rICR_PC_CB->ICR_PC_CB,
sICR_PC_CB|rICR_PC_CB->ICR_PC_CB,
sIPCBroadcast|IPC_Broadcast -> IPC_CB,
sIPCBroadcast|IPC_Broadcast -> IPC_CB,
sIPCBroadcast|IPC_Broadcast -> IPC_Broadcast,
sMsghandler | Msghandler -> Maghandler,
scommandhandler | rcommandhandler -> commandhandler },

ControlPC(PC_Off) ||
GeoPC(PC_Off) ||
NormalPC(1,PC_Off) ||
NormalPC(2,PC_Off) ||
NormalPC(3,PC_Off) ||
NormalPC(4,PC_Off) ||
NormalPC(5,PC_Off) ||
Handler ||
PDU_State_Machine(PDU_Off,false,false,false,0)));
init System;

F Poll model

sort Command = struct PDUswitchOn | PDUswitchOff | powerOn | powerOff | forcedPowerOff |
emergencyOff | onPressed | queryCRPCStatus | queryCRPCPowerOffFlag |
queryGeoStopFlag | queryGeoPCstatus | queryPCstatus ;
sort TimerData = struct pollGeoPC | pollCRPC | poll1PC;
Bmsg = struct restart | shutdown ;
PCState = struct PC_Off | PC_On | WaitingShutdown | OS_Shutdown | StopPressed;
RetVal = struct IsOperational | IsNotOperational | IsOn | IsOff | StopIsPressed | StopIsNotPressed;
PDUSate = struct PDU_Off | StartingUpCrPC | StartingUpAllPCs | System_Standby | System_On |
Emergency_Off | System_Off | Geo_Stop | none | WaitingCRPCReply |
WaitingGeoPCstatusReply | CheckingGeoStopStatus | CheckingCRPCFlag |
WaitingGeoPCstatusReply | WaitingPC1statusReply | WaitingPC2statusReply |
WaitingPC3statusReply | WaitingPC4statusReply | WaitingPCstatusReply;
IndicationMsg = struct startingUp | off | systemOn | systemStandby | geoStop ;
PCs = struct CRPC | GeoPC | NormalPC ;
act Illegal,
release,rrelease,release;
IPDU,
sICR_PC,rICR_PC,ICR_PC,
IGeoPC,rIGeoPC,IGeoPC,
scommandhandler, rcommandhandler, commandhandler : Command;
IPDUTimer: TimerData;
sICR_PCrVal,rICR_PCrVal,ICR_PCrVal,
sIGeoPCrVal,rIGeoPCrVal,IGeoPCrVal : RetVal;
sICR_PCrVal,rICR_PCrVal,ICR_PCrVal : Pos # RetVal;
IndicationCB : IndicationMsg;

proc ControlPC(s:PCState) = (
    (s==PC_Off) -> (rICR_PC(powerOn) . ControlPC(PC_On) + rICR_PC(powerOff) . Illegal . delta + rICR_PC(queryCRPCStatus).Illegal . delta + rICR_PC(queryCRPCPowerOffFlag).Illegal . delta + rICR_PC_Broadcast(restart) . Illegal . delta + rICR_PC_Broadcast(shutdown) . Illegal . delta )
rICR_PC(queryCRPCPowerOffFlag).sICR_PCrVal(IsOff).ControlPC(PC_On)+
rICR_PC_Broadcast(restart).ControlPC(PC_On) +
rICR_PC_Broadcast(shutdown).ControlPC(OS_Shutdown)
)
+
(s==OS_Shutdown) -> {
    rICR_PC(powerOn).Illegal . delta +
    rICR_PC(powerOff).ControlPC(PC_Off) +
    rICR_PC(queryCRPCPowerOffFlag).sICR_PCrVal(IsOff).ControlPC(OS_Shutdown) +
rICR_PC_Broadcast(restart).ControlPC(OS_Shutdown) +
rICR_PC_Broadcast(shutdown).ControlPC(OS_Shutdown)
}

proc GeoPC(s:PCState) = {
    (s==PC_Off) -> {
        rIGeoPC(powerOn).GeoPC(PC_On) +
        rIGeoPC(powerOff).Illegal . delta +
        rIGeoPC(queryGeoStopFlag).Illegal . delta +
        rIGeoPC(queryGeoPCstatus).sIGeoPCrVal(IsOperational).GeoPC(PC_On) +
rIGeoPC(queryGeoPCstatus).sIGeoPCrVal(IsNotOperational).GeoPC(PC_Off) +
rIGeoPC(onPressed).Illegal . delta +
        rIGeoPC_Broadcast(shutdown).GeoPC(OS_Shutdown) +
rIGeoPC_Broadcast(restart).Illegal . delta +
    }
    +
    (s==PC_On) -> {
        rIGeoPC(powerOn).Illegal . delta +
        rIGeoPC(powerOff).GeoPC(PC_Off) +
        rIGeoPC(queryGeoStopFlag).sIGeoPCrVal(StopIsNotPressed).GeoPC(PC_Off) +
rIGeoPC(queryGeoStopFlag).sIGeoPCrVal(StopIsPressed).GeoPC(StopPressed) +
rIGeoPC(queryGeoPCstatus).sIGeoPCrVal(IsOperational).GeoPC(PC_On) +
rIGeoPC(queryGeoPCstatus).sIGeoPCrVal(IsNotOperational).GeoPC(PC_Off) +
rIGeoPC(onPressed).Illegal . delta +
        rIGeoPC_Broadcast(shutdown).GeoPC(OS_Shutdown) +
rIGeoPC_Broadcast(restart).Illegal . delta +
    }
    +
    (s==OS_Shutdown) -> {
        rIGeoPC(powerOn).Illegal . delta +
        rIGeoPC(powerOff).GeoPC(PC_Off) +
        rIGeoPC(queryGeoStopFlag).Illegal . delta +
        rIGeoPC(queryGeoPCstatus).sIGeoPCrVal(IsOperational).GeoPC(OS_Shutdown) +
rIGeoPC(queryGeoPCstatus).sIGeoPCrVal(IsNotOperational).GeoPC(PC_Off) +
rIGeoPC(onPressed).Illegal . delta +
        rIGeoPC_Broadcast(shutdown).GeoPC(OS_Shutdown) +
rIGeoPC_Broadcast(restart).GeoPC(OS_Shutdown)
    }
    +
    (s==StopPressed) -> {
        rIGeoPC(powerOn).Illegal . delta +
        rIGeoPC(powerOff).GeoPC(PC_Off) +
        rIGeoPC(queryGeoStopFlag).sIGeoPCrVal(StopIsNotPressed).GeoPC(StopPressed) +
rIGeoPC(queryGeoStopFlag).sIGeoPCrVal(StopIsPressed).GeoPC(StopPressed) +
rIGeoPC(queryGeoPCstatus).sIGeoPCrVal(IsOperational).GeoPC(PC_Off) +
rIGeoPC(queryGeoPCstatus).sIGeoPCrVal(IsNotOperational).GeoPC(PC_Off) +
rIGeoPC(onPressed).GeoPC(StopPressed) +
        rIGeoPC_Broadcast(shutdown).GeoPC(OS_Shutdown) +
rIGeoPC_Broadcast(restart).GeoPC(StopPressed)
    }
};

proc NormalPC(id:Pos,s:PCState) = {
    (s==PC_Off) -> {
        rIPC(id,powerOn).NormalPC(id,PC_On) +
        rIPC(id,powerOff).Illegal . delta +
        rIPC(id,queryPCstatus).Illegal . delta +
        rIPC_Broadcast(id,shutdown).Illegal . delta +
        rIPC_Broadcast(id,restart).Illegal . delta
    }
};
(s==PC_On) -> {
  rIPC(id,powerOn) . Illegal . delta +
  rIPC(id,powerOff) . NormalPC(id,PC_Off) +
  rIPC(id,queryPCstatus) . sIPCrVal(id,IsOperational) . NormalPC(id,PC_On) +
  rIPC(id,queryPCstatus) . sIPCrVal(id,IsNotOperational) . NormalPC(id,PC_Off) +
  rIPC_Broadcast(id,shutdown) . NormalPC(id,OS_Shutdown) +
  rIPC_Broadcast(id,restart) . NormalPC(id,PC_On)
}

+ (s==OS_Shutdown) -> {
  rIPC(id,powerOn) . Illegal . delta +
  rIPC(id,powerOff) . NormalPC(id,PC_Off) +
  rIPC(id,queryPCstatus) . sIPCrVal(id,IsNotOperational) . NormalPC(id,OS_Shutdown) +
  rIPC_Broadcast(id,shutdown) . NormalPC(id,OS_Shutdown) +
  rIPC_Broadcast(id,restart) . NormalPC(id,OS_Shutdown)
}

); % the PDU design
proc PDU_State_Machine(s:PDUSState,pRPCstarted,geoPCstarted,geoPressed:Bool,state:PDUSState) = {
  (s==PDU_Off) -> {
    IPDU(PDUswitchOn) . sICR_PC(powerOn) . IndicationCB(startingUp) .
    PDU_State_Machine(StartingUpCrPC,false,false,false,none)
  }
  +
  (s==StartingUpCrPC) -> {
    IPDU(PDUswitchOff) . sICR_PC(powerOff) .
    PDU_State_Machine(PDU_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
    IPDU(powerOn) . PDU_State_Machine(StartingUpCrPC,cRPCstarted,geoPCstarted,geoPressed,state) +
    IPDU(powerOff) . PDU_State_Machine(System_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
    IPDU(forcedPowerOff) . sICR_PC(powerOff) . IndicationCB(off) .
    PDU_State_Machine(System_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
    IPDU(emergencyOff) . sICR_PC(powerOff) .
    PDU_State_Machine(Emergency_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
    IPDU(pollPC) . sICR_PC(queryCRPCStatus) .
    PDU_State_Machine(WaitingCRPCReply,cRPCstarted,geoPCstarted,geoPressed,state)
  }
  +
  (s==System_Off) -> {
    IPDU(PDUswitchOff) . PDU_State_Machine(PDU_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
    IPDU(powerOn) . sICR_PC(powerOn) . sGeoPC(powerOn) . scommandhandler(powerOn) . rrelease .
    IndicationCB(startingUp) . PDU_State_Machine(StartingUpAllPCs,false,false,false,none) +
    IPDU(powerOff) . PDU_State_Machine(System_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
    IPDU(forcedPowerOff) . PDU_State_Machine(System_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
    IPDU(emergencyOff) .
    PDU_State_Machine(Emergency_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
    IPDU(emergencyOff) . PDU_State_Machine(Emergency_Off,cRPCstarted,geoPCstarted,geoPressed,state)
  }
  +
  (s==Emergency_Off) -> {
    IPDU(PDUswitchOff) . PDU_State_Machine(PDU_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
    IPDU(powerOn) . sICR_PC(powerOn) . sGeoPC(powerOn) . scommandhandler(powerOn) . rrelease .
    IndicationCB(startingUp) . PDU_State_Machine(StartingUpAllPCs,false,false,false,none) +
    IPDU(powerOff) . PDU_State_Machine(Emergency_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
    IPDU(forcedPowerOff) .
    PDU_State_Machine(Emergency_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
    IPDU(emergencyOff) . PDU_State_Machine(Emergency_Off,cRPCstarted,geoPCstarted,geoPressed,state)
  }
  +
  (s==WaitingCRPCReply) -> {
    rICR_PCrVal(IsOperational) . IndicationCB(systemStandby) .
    PDU_State_Machine(System_Standby,true,geoPCstarted,geoPressed,state) +
    rICR_PCrVal(IsNotOperational) .
    PDU_State_Machine(StartingUpCrPC,false,geoPCstarted,geoPressed,state)
  }
  +
  (s==StartingUpAllPCs) -> {
    IPDU(PDUswitchOff) . sICR_PC(powerOff) . sGeoPC(powerOff) . scommandhandler(powerOff) .
  }
rrelease . PDU_State_Machine(PDU_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
IPDU(powerOn) .

PDU_State_Machine(StartingUpAllPCs,cRPCstarted,geoPCstarted,geoPressed,state) +
(cRPCstarted) -> IPDU(powerOff) . sGeoPC_Broadcast(shutdown) . sMsgHandler(shutdown) .
rrelease . sGeoPC(powerOff) . sCommandHandler(powerOff) . rrelease .
IndicationCB(systemStandby) .
PDU_State_Machine(System_Standby,cRPCstarted,geoPCstarted,false,none) +
(!cRPCstarted) -> IPDU(powerOff) . sGeoPC_Broadcast(shutdown) . sMsgHandler(shutdown) .
rrelease . sGeoPC(powerOff) . sCommandHandler(powerOff) . rrelease .
PDU_State_Machine(System_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
IPDU(emergencyOff) . sICR_PC(powerOff) . sGeoPC(powerOff) . sCommandHandler(powerOff) .
rrelease . IndicationCB(off) .
PDU_State_Machine(Emergency_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
IPDU(forcedPowerOff) . sICR_PC(powerOff) . sGeoPC(powerOff) . sCommandHandler(powerOff) .
rrelease . IndicationCB(off) .
PDU_State_Machine(PDU_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
IPDU(powerOff) .
PDU_State_Machine(StartingUpAllPCs,cRPCstarted,geoPCstarted,geoPressed,state) +
(cRPCstarted) -> IPDU(powerOff) . sGeoPC_Broadcast(shutdown) . sMsgHandler(shutdown) .
rrelease . sGeoPC(powerOff) . sCommandHandler(powerOff) . rrelease .
PDU_State_Machine(System_Standby,cRPCstarted,geoPCstarted,false,none) +
IPDU(powerOff) . sICR_PC(powerOff) . IndicationCB(off) .
PDU_State_Machine(System_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
IPDU(emergencyOff) . sICR_PC(powerOff) . IndicationCB(off) .
PDU_State_Machine(Emergency_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
IPDU(forcedPowerOff) . sICR_PC(powerOff) . sCommandHandler(powerOff) .
rrelease . IndicationCB(off) .
PDU_State_Machine(System_Standby,cRPCstarted,geoPCstarted,false,none) +
IPDU(powerOff) .
PDU_State_Machine(System_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
IPDU(emergencyOff) . sICR_PC(powerOff) . sGeoPC(powerOff) . sCommandHandler(powerOff) .
rrelease . IndicationCB(off) .
PDU_State_Machine(PDU_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
IPDU(powerOn) . sGeoPC(powerOn) . sCommandHandler(powerOn) . rrelease .
IndicationCB(startingUp) .
PDU_State_Machine(StartingUpAllPCs,cRPCstarted,geoPCstarted,geoPressed,state) +
IPDU(powerOff) .
PDU_State_Machine(System_Standby,cRPCstarted,geoPCstarted,geoPressed,state) +
IPDU(powerOff) . sGeoPC(powerOff) . sCommandHandler(powerOff) .
IndicationCB(systemStandby) .
(state == Geo_Stop) -> rICR_PCrVal(IsOff).

PDU_State_Machine(System_Off, cRPCstarted, geoPCstarted, geoPressed, state) +

(state == System_On) -> rICR_PCrVal(IsOff).

PDU_State_Machine(System_Off, cRPCstarted, geoPCstarted, geoPressed, none) +

(state == StartingUpAllPCs) -> rICR_PCrVal(IsOff).

PDU_State_Machine(System_On, cRPCstarted, geoPCstarted, geoPressed, none) +

(state == System_Standby) -> rICR_PCrVal(IsOff).

PDU_State_Machine(System_Standby, cRPCstarted, geoPCstarted, geoPressed, none)

) +

(s == WaitingGeoPCStatusReply) -> (rICrVal(IsOperational). sIPC(1, queryPCstatus).

PDU_State_Machine(WaitingPC1statusReply, cRPCstarted, true, geoPressed, state) +

rICrVal(IsNotOperational).

PDU_State_Machine(StartingUpAllPCs, cRPCstarted, false, geoPressed, state)

) +

(s == Geo_Stop) -> (IPDU(PDUswitchOff). sICR_PC(powerOff). sIGeoPC(powerOff). scmcommandhandler(powerOff).

release.

PDU_State_Machine(PDU_Off, cRPCstarted, geoPCstarted, geoPressed, state) +

IPDU(powerOn). sIGeoPC(onPressed). IndicationCB(systemOn).

PDU_State_Machine(System_On, cRPCstarted, geoPCstarted, geoPressed, state) +

IPDU(powerOff). sIGeoPC_Broadcast(shutdown). smMsgHandler(shutdown). rrelease.

sICR_PC(powerOff).

scmcommandhandler(powerOff). rrelease. IndicationCB(systemStandby).

PDU_State_Machine(System_Standby, cRPCstarted, false, false, state) +

IPDU(forcedPowerOff). sICR_PC(powerOff). sIGeoPC(powerOff). scmcommandhandler(powerOff).

release.

IndicationCB(off).

PDU_State_Machine(System_Off, cRPCstarted, geoPCstarted, geoPressed, state) +

IPDU(emergencyOff). sICR_PC(powerOff). sIGeoPC(powerOff). scmcommandhandler(powerOff).

release.

IndicationCB(off).

PDU_State_Machine(System_Off, cRPCstarted, geoPCstarted, geoPressed, state) +

IPDU(forcedPowerOff). sICR_PC(powerOff). sIGeoPC(powerOff). scmcommandhandler(powerOff).

release.

IPDU(forcedPowerOff). sICR_PC(powerOff). sIGeoPC(powerOff). scmcommandhandler(powerOff).

release.

IPDU(emergencyOff). sICR_PC(powerOff). sIGeoPC(powerOff). scmcommandhandler(powerOff).

release.

IPDU(emergencyOff). sICR_PC(powerOff). sIGeoPC(powerOff). scmcommandhandler(powerOff).

release.

IPDU(emergencyOff). sICR_PC(powerOff). sIGeoPC(powerOff). scmcommandhandler(powerOff).

release.

IPDU(emergencyOff). sICR_PC(powerOff). sIGeoPC(powerOff). scmcommandhandler(powerOff).

release.

IPDU(emergencyOff). sICR_PC(powerOff). sIGeoPC(powerOff). scmcommandhandler(powerOff).

release.
rIPCrVal(1,IsOperational) . sIPC(2,queryPCstatus) .
PDU_State_Machine(WaitingPC2statusReply,cRPCstarted,geoPCstarted,geoPressed,state) +
rIPCrVal(1,IsNotOperational) .
PDU_State_Machine(StartingUpAllPCs,cRPCstarted,geoPCstarted,geoPressed,state)
)
+
(s==WaitingPC2statusReply) -> (rIPCrVal(2,IsOperational) . sIPC(3,queryPCstatus) .
PDU_State_Machine(WaitingPC3statusReply,cRPCstarted,geoPCstarted,geoPressed,state) +
rIPCrVal(2,IsNotOperational) .
PDU_State_Machine(StartingUpAllPCs,cRPCstarted,geoPCstarted,geoPressed,state)
)
+
(s==WaitingPC3statusReply) -> (rIPCrVal(3,IsOperational) . sIPC(4,queryPCstatus) .
PDU_State_Machine(WaitingPC4statusReply,cRPCstarted,geoPCstarted,geoPressed,state) +
rIPCrVal(3,IsNotOperational) .
PDU_State_Machine(StartingUpAllPCs,cRPCstarted,geoPCstarted,geoPressed,state)
)
+
(s==WaitingPC4statusReply) -> (rIPCrVal(4,IsOperational) . sIPC(5,queryPCstatus) .
PDU_State_Machine(WaitingPC5statusReply,cRPCstarted,geoPCstarted,geoPressed,state) +
rIPCrVal(4,IsNotOperational) .
PDU_State_Machine(StartingUpAllPCs,cRPCstarted,geoPCstarted,geoPressed,state)
)
+
(!geoPressed) -> rIPCrVal(5,IsOperational) . IndicationCB(systemOn) .
PDU_State_Machine(System_On,cRPCstarted,geoPCstarted,geoPressed,state) +
(geoPressed) -> rIPCrVal(5,IsOperational) . IndicationCB(geoStop) .
PDU_State_Machine(Geo_Stop,cRPCstarted,geoPCstarted,geoPressed,state) +
rIPCrVal(5,IsNotOperational) .
PDU_State_Machine(StartingUpAllPCs,cRPCstarted,geoPCstarted,geoPressed,state)
);

proc Handler = sum c:Command . rcommandhandler(c) | sIPC(1,c) . sIPC(2,c) . sIPC(3,c) .sIPC(4,c) .
srelease | sIPC(5,c) . Handler +
sum m:Bmsg . rMsghandler(m) | sIPC_Broadcast(1,m) . sIPC_Broadcast(2,m) .
sIPC_Broadcast(3,m) . sIPC_Broadcast(4,m) . srelease|sIPC_Broadcast(5,m) . Handler;

proc System= hide({ IPDUtimer,ICR_PC,ICR_PC_Broadcast,
 GeoPC,IGeoPC_Broadcast,
 IPC,IPC_Broadcast,
 IPCrVal,ICr_PCrVal,IGeoPCrVal,IPCrVal ,
 release,Msghandler,commandhandler

 allow({IPDU,IPDUtimer,
 ICR_PC,ICR_PC_Broadcast,
 IGeoPC,IGeoPC_Broadcast,
 IPC ,IPC_Broadcast ,
 ICR_PCrVal,
 IGeoPCrVal,
 IPCrVal ,
 IndicationCB,
 commandhandler|IPC,Msghandler|IPC_Broadcast,
 release|IPC,
 release|IPC_Broadcast,
 release,
 Illegal

 },
 comm({ICR_PC | rICR_PC -> ICR_PC,
G. Poll model with global synchronous communication

```
sICR_PC_Broadcast | rICR_PC_Broadcast -> ICR_PC_Broadcast,
sIGeoPC | rIGeoPC -> IGeoPC,
sIGeoPC_Broadcast | rIGeoPC_Broadcast -> IGeoPC_Broadcast,
srelease | rrelease -> release,
rIPC | sIPC -> IPC,
rIPC_Broadcast | sIPC_Broadcast -> IPC_Broadcast,
sMsghandler | rMsghandler -> Msghandler,
scommandhandler | rcommandhandler -> commandhandler,
sICR_PCrVal | rICR_PCrVal -> ICR_PCrVal,
sIGeoPCrVal | rIGeoPCrVal -> IGeoPCrVal,
sIPCrVal | rIPCrVal -> IPCrVal

G Poll model with global synchronous communication

sort Command = struct PDUswitchOn | PDUswitchOff | powerOn | powerOff | forcedPowerOff | emergencyOff | onPressed | queryCRPCstatus | queryCRPCpowerOffFlag | queryGeoStopFlag | queryGeoPCstatus | queryPCstatus;

sort TimerData = struct pollGeoPC | pollCRPC | pollPC;

Bmsg = struct restart | shutdown;

PCState = struct PC_Off | PC_On | WaitingShutdown | OS_Shutdown | StopPressed;

RetVal = struct IsOperational | IsNotOperational | IsOn | IsOff | StopIsPressed | StopIsNotPressed;

PDUState = struct PDU_Off | StartingUpCrPC | StartingUpAllPCs | System_Standby | System_On | Emergency_Off | System_Off | Geo_Stop | none | WaitingCRPCReply | WaitingCRPCStatusReply | CheckingGeoStopStatus | CheckingCRPCFlag | WaitingPC1statusReply | WaitingPC2statusReply | WaitingPC3statusReply | WaitingPC4statusReply | WaitingPC5statusReply;

IndicationMsg = struct startingUp | off | systemOn | systemStandby | geoStop;

PCs = struct CRPC | GeoPC | NormalPC;

act Illegal,

srelease, rrelease, release;

IPDU,
sICR_PC, rICR_PC, ICR_PC,
sGeoPC, rGeoPC, IGeoPC,
scommandhandler, rcommandhandler, commandhandler : Command;

IPDUTimer: TimerData;

sICR_PCrVal, rICR_PCrVal, ICR_PCrVal,
sIGeoPCrVal, rIGeoPCrVal, IGeoPCrVal: RetVal;
sIPCrVal, rIPCrVal, IPCrVal: Pos # RetVal;

sIPC, rIPC, IPC: Pos # Command;

sICR_PC_Broadcast, rICR_PC_Broadcast, ICR_PC_Broadcast,
sIGeoPC_Broadcast, rIGeoPC_Broadcast, IGeoPC_Broadcast,
sMsghandler, rMsghandler, Msghandler: Bmsg;
sIPC_Broadcast, rIPC_Broadcast, IPC_Broadcast : Pos # Bmsg;
IndicationCB : IndicationMsg;
proc ControlPC(s:PCState) = ( 
  (s==PC_Off) -> ( 
    rICR_PC(powerOn) . ControlPC(PC_On) + 
    rICR_PC(powerOff) . Illegal . delta + 
    rICR_PC(queryCRPCStatus) . Illegal . delta + 
    rICR_PC_Broadcast(restart) . Illegal . delta + 
    rICR_PC_Broadcast(shutdown) . Illegal . delta 
  )
  + 
  (s==PC_On) -> ( 
    rICR_PC(powerOn) . Illegal . delta + 
    rICR_PC(powerOff) . ControlPC(PC_Off) + 
    rICR_PC(queryCRPCStatus) . sICR_PCrVal(IsOperational) . ControlPC(PC_On) + 
    rICR_PC(queryCRPCPowerOffFlag) . sICR_PCrVal(IsOn) . ControlPC(PC_On) + 
    rICR_PC_Broadcast(restart) . ControlPC(PC_On) + 
    rICR_PC_Broadcast(shutdown) . ControlPC(OS_Shutdown) 
  )
  + 
  (s==OS_Shutdown) -> ( 
    rICR_PC(powerOn) . Illegal . delta + 
    rICR_PC(powerOff) . GeoPC(PC_Off) + 
    rICR_PC(queryCRPCStatus) . Illegal . delta + 
    rICR_PC_Broadcast(restart) . GeoPC(OS_Shutdown) 
  );
proc GeoPC(s:PCState) = ( 
  (s==PC_Off) -> ( 
    rIGeoPC(powerOn) . GeoPC(PC_On) + 
    rIGeoPC(powerOff) . Illegal . delta + 
    rIGeoPC(queryGeoStopFlag) . Illegal . delta + 
    rIGeoPC(queryGeoPCstatus) . Illegal . delta + 
    rIGeoPC(onPressed) . Illegal . delta + 
    rIGeoPC_Broadcast(shutdown) . Illegal . delta + 
    rIGeoPC_Broadcast(restart) . Illegal . delta 
  )
  + 
  (s==PC_On) -> ( 
    rIGeoPC(powerOn) . Illegal . delta + 
    rIGeoPC(powerOff) . GeoPC(PC_Off) + 
    rIGeoPC(queryGeoStopFlag) . sIGeoPCrVal(StopIsNotPressed) . GeoPC(PC_On) + 
    rIGeoPC(queryGeoPCstatus) . Illegal . delta + 
    rIGeoPC_Broadcast(shutdown) . GeoPC(OS_Shutdown) + 
    rIGeoPC_Broadcast(restart) . GeoPC(PC_On) 
  )
  + 
  (s==OS_Shutdown) -> ( 
    rIGeoPC(powerOn) . Illegal . delta + 
    rIGeoPC(powerOff) . GeoPC(PC_Off) + 
    rIGeoPC(queryGeoStopFlag) . Illegal . delta + 
    rIGeoPC(queryGeoPCstatus) . sIGeoPCrVal(IsNotOperational) . GeoPC(OS_Shutdown) + 
    rIGeoPC(onPressed) . Illegal . delta + 
  );
rIGeoPC_Broadcast(shutdown) . GeoPC(OS_Shutdown) +

rIGeoPC_Broadcast(restart) . GeoPC(OS_Shutdown)
)

(s==StopPressed) -> (  

rIGeoPC(powerOn) . Illegal . delta +  

rIGeoPC(powerOff) . GeoPC(PC_Off) +  

rIGeoPC(queryGeoStopFlag) . sIGeoPCrVal(StopIsPressed) . GeoPC(StopPressed) +  

rIGeoPC(queryGeoPCstatus) . sIGeoPCrVal(IsOperational) . GeoPC(StopPressed) +  

rIGeoPC(onPressed) . GeoPC(PC_On) +  

rIGeoPC_Broadcast(shutdown) . GeoPC(OS_Shutdown) +  

rIGeoPC_Broadcast(restart) . GeoPC(StopPressed)
)


proc NormalPC(id:Pos,s:PCState) = (  

(s==PC_Off) -> (  

rIPC(id,powerOn) . NormalPC(id,PC_On) +  

rIPC(id,powerOff) . Illegal . delta +  

rIPC(id,queryPCstatus) . Illegal . delta +  

rIPC_Broadcast(id,shutdown) . Illegal . delta +  

rIPC_Broadcast(id,restart) . Illegal . delta
)

+  

(s==PC_On) -> (  

rIPC(id,powerOn) . Illegal . delta +  

rIPC(id,powerOff) . NormalPC(id,PC_Off) +  

rIPC(id,queryPCstatus) . sIPCrVal(id,IsOperational) . NormalPC(id,PC_On) +  

rIPC(id,queryPCstatus) . sIPCrVal(id,IsNotOperational) . NormalPC(id,PC_On) +  

rIPC_Broadcast(id,shutdown) . NormalPC(id,OS_Shutdown) +  

rIPC_Broadcast(id,restart) . NormalPC(id,PC_On)
)

+  

(s==OS_Shutdown) -> (  

rIPC(id,powerOn) . Illegal . delta +  

rIPC(id,powerOff) . NormalPC(id,PC_Off) +  

rIPC(id,queryPCstatus) . sIPCrVal(id,IsNotOperational) . NormalPC(id,OS_Shutdown) +  

rIPC_Broadcast(id,shutdown) . NormalPC(id,OS_Shutdown) +  

rIPC_Broadcast(id,restart) . NormalPC(id,OS_Shutdown)
)

);  

proc PDU_State_Machine(s:PDUState,cRPCstarted,geoPCstarted,geoPressed:Bool,state:PDUState) = (  

(s==PDU_Off) -> (  

IPDU(PDUswitchOn) . sICR_PC(powerOn) . IndicationCB(startingUp) .

PDU_State_Machine(StartingUpCrPC,false,false,false,none)
)

+  

(s==StartingUpCrPC) -> (  

IPDU(PDUswitchOff) . sICR_PC(powerOff) .  

PDU_State_Machine(StartingUpAllPCs,false,false,false,none) +  

IPDU(powerOn) . PDU_State_Machine(StartingUpCrPC,cRPCstarted,geoPCstarted,geoPressed,state) +  

IPDU(powerOff) . PDU_State_Machine(StartingUpCrPC,cRPCstarted,geoPCstarted,geoPressed,state) +  

IPDU(forcedPowerOff) . sICR_PC(powerOff) . PDU_State_Machine(System_Off,cRPCstarted,geoPCstarted,geoPressed,state) +  

IPDU(emergencyOff) . sICR_PC(powerOff) . PDU_State_Machine(Emergency_Off,cRPCstarted,geoPCstarted,geoPressed,state) +  

IPDUTimer(pollPC) . sICR_PC(queryPCstatus) . PDU_State_Machine(WaitingCRPCReply,cRPCstarted,geoPCstarted,geoPressed,state)
)

+  

(s==System_Off) -> (  

IPDU(PDUswitchOff) . PDU_State_Machine(PDU_Off,cRPCstarted,geoPCstarted,geoPressed,state) +  

IPDU(powerOn) . sICR_PC(powerOn) . sGeoPC(powerOn) . commandhandler(powerOn) .

IndicationCB(startingUp) . PDU_State_Machine(StartingUpAllPCs,false,false,false,none) +
IPDU(powerOff) . PDU_State_Machine(System_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
IPDU(forcedPowerOff) . PDU_State_Machine(System_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
IPDU(emergencyOff) . PDU_State_Machine(System_Off,cRPCstarted,geoPCstarted,geoPressed,state)
)
*
(s==Emergency_Off) ->
IPDU(PDUswitchOff) . PDU_State_Machine(PDU_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
IPDU(powerOn) . sICR_PC(powerOn) . sGeoPC(powerOn) | scommandhandler(powerOn) .
IndicationCB(startingUp) . PDU_State_Machine(StartingUpAllPCs,false,geoPCstarted,geoPressed,state) +
IPDU(powerOff) . PDU_State_Machine(Emergency_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
IPDU(forcedPowerOff) . PDU_State_Machine(Emergency_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
IPDU(emergencyOff) . PDU_State_Machine(Emergency_Off,cRPCstarted,geoPCstarted,geoPressed,state)
)
*
(s==WaitingCRPCReply) ->
 rICR_PCrVal(IsOperational) . IndicationCB(systemStandby) .
PDU_State_Machine(System_Standby,true,geoPCstarted,geoPressed,state) +
rICR_PCrVal(IsNotOperational) . PDU_State_Machine(StartingUpCrPC,false,geoPCstarted,geoPressed,state)
)
*
(s==StartingUpAllPCs) ->
IPDU(PDUswitchOff) . sICR_PC(powerOff) . sGeoPC(powerOff) | scommandhandler(powerOff) .
PDU_State_Machine(PDU_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
IPDU(powerOn) . PDU_State_Machine(System_Standby,cRPCstarted,geoPCstarted,geoPressed,startingUp) +
IPDU(powerOff) . PDU_State_Machine(System_Standby,geoPCstarted,geoPressed,state) +
IPDU(forcedPowerOff) . sICR_PC(powerOff) . IndicationCB(off) .
PDU_State_Machine(System_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
IPDU(emergencyOff) . sICR_PC(powerOff) . IndicationCB(off) .
PDU_State_Machine(Emergency_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
IPDUTimer(pollCRPC) . sICR_PC(queryCRPCPowerOffFlag) .
PDU_State_Machine(CheckingCRPCFlag,cRPCstarted,geoPCstarted,geoPressed,StartingUpAllPCs) +
(!cRPCstarted) ->
IPDUTimer(pollCRPC) . PDU_State_Machine(StartingUpAllPCs,cRPCstarted,geoPCstarted,geoPressed,state)
)
*
(s==System_Standby) ->
IPDU(PDUswitchOff) . sICR_PC(powerOff) .
PDU_State_Machine(PDU_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
IPDU(powerOn) . sICR_PC(powerOn) . sGeoPC(powerOn) | scommandhandler(powerOn) .
IndicationCB(startingUp) . PDU_State_Machine(StartingUpAllPCs,false,geoPCstarted,geoPressed,state) +
IPDU(powerOff) . PDU_State_Machine(System_Standby,cRPCstarted,geoPCstarted,geoPressed,off) +
IPDU(forcedPowerOff) . sICR_PC(powerOff) . IndicationCB(off) .
PDU_State_Machine(System_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
IPDU(emergencyOff) . sICR_PC(powerOff) . IndicationCB(off) .
PDU_State_Machine(Emergency_Off,cRPCstarted,geoPCstarted,geoPressed,state) +
IPDUpowerOffFlag) . sICR_PC(queryCRPCPowerOffFlag) .
PDU_State_Machine(CheckingCRPCFlag,cRPCstarted,geoPCstarted,geoPressed,StartingUpAllPCs) +
(!cRPCstarted) ->
PDU_State_Machine(StartingUpAllPCs,cRPCstarted,geoPCstarted,geoPressed,state)
)
*
(s==WaitingCRPCStatusReply) ->
 rICR_PCrVal(IsOperational) . sGeoPC(queryCRPCStatus) .
PDU_State_Machine(WaitingCRPCStatusReply,geoPCstarted,geoPressed,startingUp) +
rICR_PCrVal(IsNotOperational) . PDU_State_Machine(System_Standby,geoPCstarted,geoPressed,startingUp) +
IPDUTimer(pollCRPC) . PDU_State_Machine(StartingUpAllPCs,cRPCstarted,geoPCstarted,geoPressed,state)
)
*
(s==CheckingGeoStopStatus) ->

(state==StartingUpAllPCs) -> rIGeoPCrVal(StopIsPressed) .
PDU_State_Machine(StartingUpAllPCs,cRPCstarted,geoPCstarted,true,state) +
(state==System_On) -> rIGeoPCrVal(StopIsPressed) . IndicationCB(geoStop) .
PDU_State_Machine(geoStop,cRPCstarted,geoPCstarted,true,state) +
(state==StartingUpAllPCs) -> rIGeoPCrVal(StopIsNotPressed) .
PDU_State_Machine(StartingUpAllPCs,cRPCstarted,geoPCstarted,false,state) +
(state==System_On) -> rIGeoPCrVal(StopIsNotPressed) .
PDU_State_Machine(System_On,cRPCstarted,geoPCstarted,false,state)
+

(s==CheckingCRPCFlag) -> {

  (state!=System_Standby) -> rICR_PCrVal(IsOn) . sICR_PC_Broadcast(shutdown) .
  sIGeoPC_Broadcast(shutdown) . sMcghandler(shutdown) . rrelease . sICR_PC(powerOff) .
  PDU_State_Machine(System_Standby,cRPCstarted,geoPCstarted,geoPressed,state) +
  (state==geoPCstarted,geoPressed,state)

  (state==StartingUpAllPCs) -> rIGeoPCrVal(StopIsNotPressed) .
  PDU_State_Machine(System_On,cRPCstarted,geoPCstarted,false,state)
+

  (s==System_Standby) . sICR_PC_Broadcast(shutdown) .
  sIGeoPC_Broadcast(shutdown) . sMcghandler(shutdown) . rrelease . sICR_PC(powerOff) .
  sIGeoPC(powerOff) | scommandhandler(powerOff) . IndicationCB(off) .
  PDU_State_Machine(System_Off,cRPCstarted,geoPCstarted,geoPressed,state) +

  (state==StartingUpAllPCs) -> rIGeoPCrVal(StopIsNotPressed) .
  PDU_State_Machine(StartingUpAllPCs,cRPCstarted,geoPCstarted,false,state) +

  (state==System_On) -> rIGeoPCrVal(StopIsNotPressed) .
  PDU_State_Machine(System_On,cRPCstarted,geoPCstarted,false,state)
+

  (s==CheckingGeoPCFlag) -> {

    (state!=System_Standby) -> rICR_PCrVal(IsOn) . sICR_PC_Broadcast(shutdown) .
    sIGeoPC_Broadcast(shutdown) . sMcghandler(shutdown) . rrelease . sICR_PC(powerOff) .
    sIGeoPC(powerOff) | scommandhandler(powerOff) . IndicationCB(off) .
    PDU_State_Machine(System_Standby,cRPCstarted,geoPCstarted,geoPressed,state) +

    (state==geoPCstarted,geoPressed,state)

  }
(s==WaitingPC1statusReply) -> (rIPCrVal(1,IsOperational) . sIPC(2,queryPCstatus) .
    PDU_State_Machine(WaitingPC2statusReply,crPCstarted,geoPCstarted,geoPressed,state) +
    rIPCrVal(1,IsNotOperational) .
    PDU_State_Machine(StartingUpAllPCs,crPCstarted,geoPCstarted,geoPressed,state)
)
+
( s==WaitingPC2statusReply) -> (rIPCrVal(2,IsOperational) . sIPC(3,queryPCstatus) .
    PDU_State_Machine(WaitingPC3statusReply,crPCstarted,geoPCstarted,geoPressed,state) +
    rIPCrVal(2,IsNotOperational) .
    PDU_State_Machine(StartingUpAllPCs,crPCstarted,geoPCstarted,geoPressed,state)
)
+
( s==WaitingPC3statusReply) -> (rIPCrVal(3,IsOperational) . sIPC(4,queryPCstatus) .
    PDU_State_Machine(WaitingPC4statusReply,crPCstarted,geoPCstarted,geoPressed,state) +
    rIPCrVal(3,IsNotOperational) .
    PDU_State_Machine(StartingUpAllPCs,crPCstarted,geoPCstarted,geoPressed,state)
)
+
( s==WaitingPC4statusReply) -> (rIPCrVal(4,IsOperational) . sIPC(5,queryPCstatus) .
    PDU_State_Machine(WaitingPC5statusReply,crPCstarted,geoPCstarted,geoPressed,state) +
    rIPCrVal(4,IsNotOperational) .
    PDU_State_Machine(StartingUpAllPCs,crPCstarted,geoPCstarted,geoPressed,state)
)
+
( s==WaitingPC5statusReply) -> (!geoPressed) -> rIPCrVal(5,IsOperational) . IndicationCB(systemOn) .
    PDU_State_Machine(System_On,crPCstarted,geoPCstarted,geoPressed,state) +
    ( geoPressed) -> rIPCrVal(5,IsOperational) . IndicationCB(geoStop) .
    PDU_State_Machine(Geo_Stop,crPCstarted,geoPCstarted,geoPressed,state) +
    rIPCrVal(5,IsNotOperational) .
    PDU_State_Machine(StartingUpAllPCs,crPCstarted,geoPCstarted,geoPressed,state)
);

proc Handler = sum c:Command . rcommandhandler(c) | sIPC(1,c) | sIPC(2,c) | sIPC(3,c) | sIPC(4,c) | sIPC(5,c) .
    Handler + sum m:Bmsg . rMsghandler(m) | sIPC_Broadcast(1,m) .
    sIPC_Broadcast(2,m) . sIPC_Broadcast(3,m) . sIPC_Broadcast(4,m) .
    release(sIPC_Broadcast(5,m)) . Handler;

proc System=
    hide({
    IPDUTimer, ICR_PC, ICR_PC_Broadcast,
    IGeoPC, IGeoPC_Broadcast,
    IPC, IPC_Broadcast,
    ICR_PCrVal,
    IGeoPCrVal,
    IPCrVal,
    release, Msghandler, commandhandler
},
    allow({
    IPDUTimer, ICR_PC, ICR_PC_Broadcast,
    IGeoPC, IGeoPC_Broadcast,
    IPC, IPC_Broadcast,
    ICR_PCrVal,
    IGeoPCrVal,
    IPCrVal,
    IndicationCB,
    IGeoPC | commandhandler | IPC | IPC | IPC | IPC | IPC,
    Msghandler | IPC_Broadcast, release | IPC_Broadcast,
    Illegal
},

\[
\text{comm}\{(\text{sICR\_PC | rICR\_PC} \to \text{ICR\_PC}, \\
\text{sICR\_PC\_Broadcast | rICR\_PC\_Broadcast} \to \text{ICR\_PC\_Broadcast}, \\
\text{sICGeoPC | rICGeoPC} \to \text{ICGeoPC}, \\
\text{sICGeoPC\_Broadcast | rICGeoPC\_Broadcast} \to \text{ICGeoPC\_Broadcast}, \\
\text{srelease | rrelease} \to \text{release}, \\
\text{rIPC | sIPC} \to \text{IPC}, \\
\text{rIPC\_Broadcast | sIPC\_Broadcast} \to \text{IPC\_Broadcast}, \\
\text{sMsghandler | rMsghandler} \to \text{Msghandler}, \\
\text{scommandhandler | rcommandhandler} \to \text{commandhandler}, \\
\text{sICR\_PCrVal | rICR\_PCrVal} \to \text{ICR\_PCrVal}, \\
\text{sICGeoPCrVal | rICGeoPCrVal} \to \text{ICGeoPCrVal}, \\
\text{sIPCrVal | rIPCrVal} \to \text{IPCrVal}\), \\
\text{ControlIPC(PC\_Off)} \mid | \\
\text{GeoPC(PC\_Off)} \mid | \\
\text{NormalIPC(1,PC\_Off)} \mid | \\
\text{NormalIPC(2,PC\_Off)} \mid | \\
\text{NormalIPC(3,PC\_Off)} \mid | \\
\text{NormalIPC(4,PC\_Off)} \mid | \\
\text{NormalIPC(5,PC\_Off)} \mid | \\
\text{Handler} \mid | \\
\text{PDU\_State\_Machine(PDU\_Off, false, false, false, none))};
\]

\text{init System;}