Analyzing a Controller of a Power Distribution Unit using Formal Methods

Jan Friso Groote
Eindhoven University of Technology
Eindhoven, The Netherlands
Email: j.f.groote@tue.nl

Ammar Osaiweran
Eindhoven University of Technology
Eindhoven, The Netherlands
Email: a.a.h.osaiweran@tue.nl

Jacco Wesselius
BU Interventional X-ray
Philips Healthcare
Best, The Netherlands
Email: jacco.wesselius@philips.com

***Abstract*—This paper reports on the steps to formally specify and verify the behavior of a controller of a power distribution unit (PDU) using the Analytical Software Design (ASD) method. The controller of the underlying PDU mainly controls the distribution of power and network messages to a number of attached PCs and devices of X-ray systems. The behavioral correctness of the controller is critical in order to provide the clinical users the expected behavior of the system. The design of the controller was thoroughly reviewed by team members but, as a result of the behavioral verification using ASD, two previously unrevealed errors were identified within the design of the PDU controller. According to the development team of the PDU the work has had a major benefit of improving the design of the controller and locating errors that would have been hard to find otherwise by traditional testing.**

I. Introduction

Philips Healthcare is a leading provider of highly sophisticated computerized X-Ray systems. These systems consist of a number of distributed devices and computers that require a reliable source of power control. The distribution of power to these components is established through a power distribution unit (PDU) attached to the main source of power in hospitals.

Users of the system interact with the PDU through an external console, which includes a number of buttons. The console is attached to an embedded controller that controls the flow of power to the components through a number of power taps. The controller communicates with the devices via a network regarding required changes to the powering aspects of the system. If the PDU does function incorrectly, components may unpredictably be with or without power when they desire, rendering the system useless or even dangerous.

Establishing the behavioral correctness of such type of systems is known to pose real challenges to conventional testing methods, due to the concurrent nature of the devices and the non-trivial interactions among them. It requires a running system in a real environment, and it takes hours or days to execute a single run. Therefore, techniques for systematic error detection and early defect prevention are encouraged, especially to reduce costs, efforts and time devoted to detecting errors and correcting them at later stages of industrial projects.

The above necessities motivate this work, which in particular aims at formal modeling and verifying the behavior of the PDU controller and the surrounding devices on the boundary using the Analytical Software Design approach [1], [8], [14]. In particular, we exploit a number of formal techniques provided by ASD, namely the ASD specification and the behavioral verification using model checking. ASD has been applied to the development of larger systems, and some reports regarding its application to industrial control software at Philips Healthcare are available [8], [1], [7].

Throughout this paper we illustrate the verification steps of a typical system as a guide for others who also wish to apply formal verification. We further exploit the case study to elaborate more on how the ASD technology was used for specifying and verifying components compositionally, following predefined steps, namely by considering first the external behavior and then slowly designing the internals of components matching the predefined external behavior.

The design of the PDU controller was thoroughly reviewed by team members [11], [10], and then was a target of formal verification using ASD, exactly at the phase where designers and architects had explored various design alternatives, and ready to start the actual implementation. But, the controller included two previously uncovered errors, detected via specification review and model checking, supplied by the ASD technology. This led to improvements to the design, and further increased the confidence of the correctness of the controller.

As will be demonstrated below, the case also highlights the importance of specification completeness, not only to provide precise complete specification as input for team review processes but also to increase the chance of detecting veiled errors before and after performing formal behavioral verification using model checking.

The specification completeness, specification review and formal behavioral verification provided a key benefit by easily locating design errors in the PDU controller that would be hard to find through conventional testing.

This paper is organized as follows. Section II introduces the context of the PDU controller. In Section III we give an overview of the ASD method to the extent needed in this paper. Our experimental method of modeling and verifying the PDU controller is demonstrated in Section IV, where we further explain the unveiled errors and how precisely they had been discovered. The efforts of modeling and verifying the PDU controller are described in Section V.

II. The design description

We provide an overview of the design context, hardware and software components, and the signals exchanged through the system. Figure 1 depicts the structure of an X-ray system that comprises a number of distributed devices and PCs connected to a Power Distribution Unit.

The PDU is attached to an external power source, via a mains switch. The PDU is responsible for distributing power and related communication signals to the attached devices and PCs. Below we detail these components to the extent related to this work.

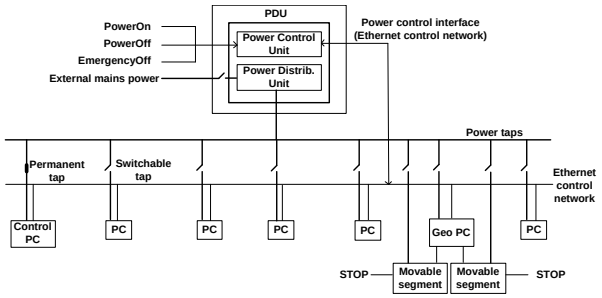

Fig. 1. Power, network, and device distribution

The Power Distribution Unit. The PDU interacts with its external users by a console, which contains a number of buttons: *PowerOn*, *PowerOff*, and *EmergencyOff*. The PDU also comprises a base module that hosts a number of power taps. It further houses internal units: a Power Control Unit, which controls the flow of network messages, and a Power Distribution Unit, which controls the distribution of power (Figure 1).

Through switching the power taps the PDU controls the flow of power to the devices and the PCs. The type of power taps can either be switchable or permanent. The switchable taps can potentially be switched on/off by the PDU upon requests of external users, issued by pressing the buttons. For example, when the system is operational and a user presses the *PowerOff* button for 3 seconds, all switchable taps are switched off so that all attached components are powered off except those attached to the permanent taps (powering off the system in an orderly fashion).

The permanent taps constantly supply power to a number of components that must always be up-and-running (e.g., for remote access purposes). The permanent taps can also be switched off in some special cases. For example the PDU switches off all taps when the external user presses the *PowerOff* button for 10 seconds, forcing all components to be powered off.

The PDU comprises a controller that includes a state machine for maintaining the states of the system. The state machine is introduced below.

In order to supply power to the system in case of failure of the main source of power in the field, an uninterruptible power supply (UPS) is attached to the PDU.

Devices and PCs. A number of devices and PCs are connected to the PDU, each of which has distinct responsibilities for achieving the required clinical applications. All components exhibit the same start-up and shutdown behavior (e.g., powering up, starting the operating system (OS) and the clinical applications, shutting down OS, etc.), controlled systematically by the PDU. The high-level behavior of these devices by means of state machines are introduced below.

All PCs depicted in Figure 1 are attached to switchable taps except the ControlPC which is attached to a permanent tap.

The GeoPC (Geometry PC) is responsible for controlling motorized movements of a number of movable parts, such as the table where patients can lay and the X-ray stands. The movable parts are supplied with emergency *Stop* buttons, attached to their bodies. The clinical users can press these buttons to stop any movement in order to avoid any potential damage that might occur during the motorized movements in the field. Upon pressing a *Stop* button, the GeoPC instructs the PDU to switch off the taps connected to the motor drives of the movable parts.

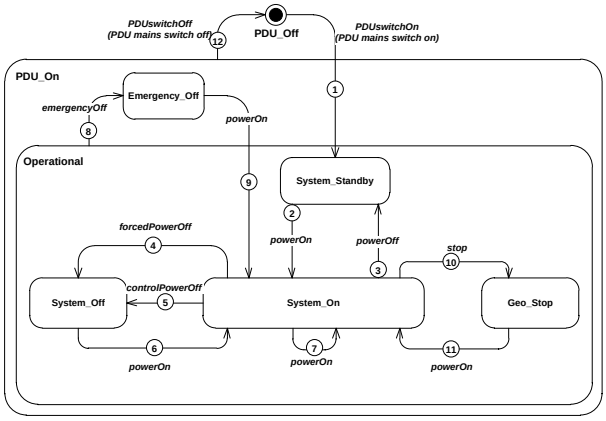

Fig. 2. The high-level behavior of the PDU [11]

External user commands. As a consequence of pressing the buttons on the user console, user commands are generated and then received by the PDU controller. Depending on the command and the state, the controller decides to send messages (introduced below) around to the devices and the PCs through the network or to switch the taps on/off.

By pressing the *PowerOn* button, a *powerOn* command is fired. Pressing the *PowerOff* button for 3 seconds generates a *powerOff* command while pushing the button for more than 10 seconds fires a *forcedPowerOff* command. The *EmergencyOff* completely cuts down any source of power (including the UPS) to the system, via an internal switch, positioned inside the PDU. The *EmergencyOff* button is pressed to ensure that the system is immediately powered off in the presence of calamities.

Internal system messages. The PDU can send and receive the following messages through the network: *shutdown*, *restart*, *controlPowerOff*, and *stop*.

The *shutdown* and *restart* are broadcast messages sent from the PDU to the PCs. The *shutdown* message instructs the devices to gradually shutdown their running applications and then the operating systems. The *restart* message requests the PCs to reboot their operating systems.

ControlPowerOff is a message issued from the ControlPC to the PDU, while *stop* is a message sent from the GeoPC to the PDU. Through the *controlPowerOff* message users of the ControlPC can instruct the PDU to systematically power off the entire system. The *stop* signal is sent by the GeoPC when any of the *Stop* buttons on the movable parts is pressed, so the PDU directly switches off all taps that supply the motor drives. The motor drives are powered on again when the user presses the *powerOn* button on the console.

Transition	Activity
1	Boot PDU; the PDU switches on all permanent power taps; the ControlPC is operational.
2	The PDU switches on all switchable taps, one by one to avoid a big inrush current; all devices are operational.
3	The PDU broadcasts a "shutdown" 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.
4	The PDU immediately switches off all power taps.
5	The PDU broadcasts a "shutdown" 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.
6	The PDU switches on all taps, one by one to avoid a big inrush current; all devices are started (all devices are operational including the ControlPC).
7	The PDU broadcasts a "restart" message; the operating systems of all control devices are restarted.
8	Disconnect the PDU internal power bus and UPS.
9	The PDU switches on all taps, one by one to avoid a big inrush current; all devices are started (all devices are operational including the ControlPC).
10	The PDU switches off the power taps that supply motor drives of movable parts.
11	The PDU switch on the power taps that supply motor drives of movable parts.
12	The PDU is switched off; all taps are switched off.

TABLE I
ACTIVITIES REQUIRED FOR EACH TRANSITION OF THE PDU STATE MACHINE [11]

The state transition diagrams. Figure 2 depicts the high-level behavior of the PDU controller from a system-level perception after implementation details are abstracted away [11]. The depicted state machine containes contains embedded errors that are corrected later. The user and system signals introduced earlier constitute stimuli and responses of the state machine. In addition to these events, the *PDUswitchOn* and *PDUswitchOff* events are used for modeling purposes to indicate switching the external mains switch on and off, respectively.

The effects of these signals on the behavior of the system differ upon the present state of the PDU. For example, when the PDU is in *System_Standby* and the *powerOn* signal is received, it switches on the switchable taps, so that the attached PCs and devices start up. But, if the PDU is in the *System_On* state and the *powerOn* signal is received, then the PDU broadcasts the *restart* message across the network. The detailed activities required for each transition of the state machine of Figure 2 are depicted in Table I.

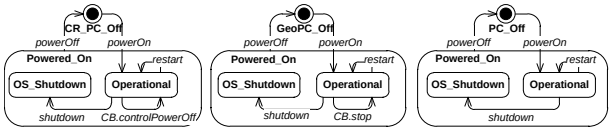

Fig. 3. The external behavior of all PCs

Here, the PDU, the devices and the PCs are assumed to be well functioning. All error handling details or recovery operations are removed from the state machine.

The state machine of Figure 2 depicts only the *stable* states of the system. The transiting states between any two stable states are excluded from the diagram. For example, when the PDU is off and then is switched on, the PDU transits to the *System_Standby* state, where the ControlPC is assumed to be successfully started and fully operational. The intermediate transiting state between the *PDU_Off* and the *System_Standby* state that ensures that the ControlPC is fully operational is removed. The same assumption applies to other states. For example, the *System_On* state implies the situation where all PCs are fully operational. All intermediate transiting states, which ensure that the PCs are fully operational, leading to *System_On* are removed.

We introduce the external behavior of the PCs with respect to the PDU. All PCs exhibit almost the same startup and shutdown behavior, see Figure 3. Initially, they are all in the *Off* state. Once a tap of a PC is switched on, the PC automatically launches its operating system and then starts up its clinical applications. When the applications are successfully started, the PC transits to the *Operational* state; this is indicated by the *powerOn* transition from the *Off* to the *Operational* states. If a PC receives a *restart* message at the *Operational* state, it restarts the operating system and the applications. But, if the *shutdown* message is received, the PC closes all running applications and shuts the operating system down.

The ControlPC and the GeoPC have two additional transitions: *CB.controlPowerOff* and *CB.stop*. They are callback (CB) events sent to the PDU, where the first indicates that the user of the ControlPC has requested the PDU to entirely power off the system, and the second indicates that the *Stop* button on a movable segment has been pressed.

The movable parts can be powered on or off by the PDU. They don't receive or send the PDU any signal through the network. The behavior of these segments is straightforward, and hence the corresponding specification is omitted (two actions of *powerOff* and *powerOn* affecting two states $Segment_x\_Off$ and $Segment_x\_On$, where $_x$ is the device id).

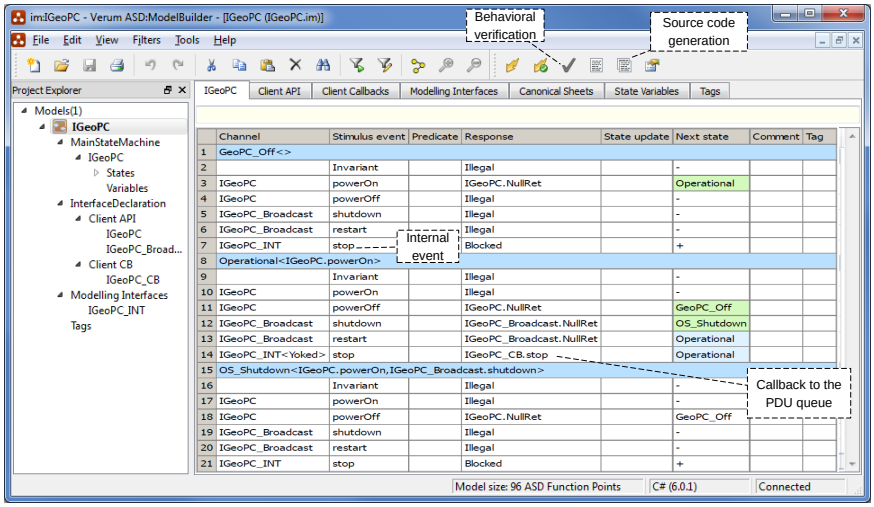

Fig. 4. The ASD interface model of the GeoPC using a tabular format

III. Analytical Software Design

We describe the Analytical Software Design (ASD) technology to the limit required for this article. ASD is a novel model-based technology that combines formal mathematical methods such as Sequence-based Specification (SBS) [2], Communicating Sequential Processes (CSP) [13] and its model checker FDR (Failures-Divergence Refinement) [4] with component based software development [3].

A. The specification of ASD models

Typical ASD components are structured in levels (e.g., distributed in a hierarchy) to facilitate constructing and verifying components in isolation. To develop any ASD component two basic types of models are required: an interface model and a design model. The interface model is created first. It precisely describes the required external behavior exposed to the clients of the component at an upper level, such that all internal interactions with used components at a lower level are not considered. The design model describes the concrete behavior including all internal interactions with used components. The description of both models is supported by a commercial tool, called the ASD ModelBuilder.

The ASD models are state machines described in a tabular format, called the SBS tables. An example of the tabular specification related to the GeoPC state machine of Figure 3 is depicted in Figure 4. As can be seen from the specification, each table is a state, where all potential input stimuli are listed in rule cases (rows of tables). A rule case comprises a number of items, such as an interface (channel), a stimulus, predicate (conditions on the stimulus), responses, state (or predicate) updates, a next state, comments, and tags of informal requirements.

The set of stimuli of a component consists of events invoked by clients located at an upper level plus callback events sent by used components at a lower level. The set of responses includes events sent to used components plus callback events sent to upper client components. Calls from client components to used components are synchronous, but callback events sent by used components to the client components are asynchronous and stored locally in a FIFO queue of the target client component.

For specification completeness, the set of user-defined responses is extended with special purpose responses: *Illegal*, *Blocked*, and *Null*. The *Illegal* response denotes that invoking a stimulus is illegal. The *Blocked* response denotes that the corresponding stimulus cannot happen. The *Null* response denotes no action is required when the stimulus is invoked: consuming a call, for instance.

In all presented models throughout this paper the *NullRet* response indicates the completion of the external request. A channel postfixed by *INT* denotes an internal channel, not visible to the client. The corresponding stimulus event of an internal channel denotes spontaneous event stimulated by the component internally, not synchronized with client components located at a higher level. A channel postfixed by CB indicates a callback stimulus/response event received/sent from/to a queue of the client.

To clarify the modeling conventions above, consider Figure 4. The *NullRet* of rule case 3 indicates that the *powerOn* event is successfully completed, and the GeoPC transits to the *Operational* state. The spontaneous *stop* stimulus of the *IGeoPC_INT* channel of rule case 14 denotes that the GeoPC stop button has internally been pressed, and as a response the GeoPC notifies the PDU controller by sending the *IGeoPC_CB.stop* callback event to the queue of the PDU.

The yoked internal channel of rule case 14 indicates that the number of allowed callbacks (listed in the corresponding response list of the rule case) in the queue is restricted. In our case study only one *stop* event and one *controlPowerOff* event are allowed to be stored in the queue of the PDU at any time. Hence, the queue can contain two messages at maximum.

B. Formal verification of ASD models

ASD components are specified and formally verified in isolation to mitigate the state space explosion problem, which may occur if all concrete components are verified at once. The specification and verification of ASD components are straightforward and performed as follows. First, the interface model of the component being developed is created, and then the interface models of the used components at a lower level are constructed. Second, the design model of the component is created such that all interactions with the used interfaces are described.

Upon the completion of the specification of ASD design and interface models, the formal behavioral verification using model checking can be established. From the tabular specification of ASD models, a translation to CSP models is obtained automatically by the ModelBuilder application. Then, the ModelBuilder constructs a combined CSP model that includes the parallel composition of the design model of the component plus the interface models of the used components. After that, the combined model is examined using the FDR model checker for the absence of deadlocks (crashes or failure to proceed with any action), livelocks (hanging due to entering an endless loop of internal events) and illegal (unexpected) calls.

When the above checks have succeeded, the combined model is examined against the interface model of the component, to formally check that the combined model matches the prescribed external specification. In other words, the combined model is checked for whether it correctly and formally refines the prescribed interface behavior. This is precisely established using the refinement checks supplied by FDR where the interface model of the component is the specification and the combined model is the implementation [4].

Note that, when the interface model is formally refined by the combined model, the interface model represents all lower level components. It can also be used during the verification of the upper level client components. The details of the step-wise verification systematically performed level-by-level according to the ASD method are outside the scope of this paper since our controller design fits in one level, but we refer to [8], [7] for further information.

All CSP models are hidden from end-users of the ModelBuilder. Although source code can also be generated automatically from the ASD models, we merely used the technology for specification and design verification purposes.

IV. Analyzing the PDU behavior

In this section our experimental methodology is sketched, summarizing the series of steps followed through the experiment of verifying the PDU design. We used the ASD ModelBuilder version 6.1.0 for describing the behavior of our components. The features supplied by this version seemed to be a good fit to our aim at modeling the PDU behavior. The formal verification using CSP and FDR is performed on a remote server located at Verum, the ASD company.

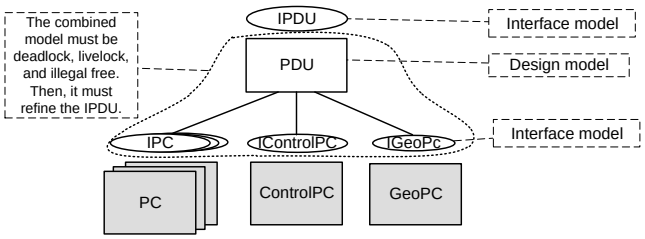

Fig. 5. The structure of ASD models

The structure of ASD models of the PDU is depicted in Figure 5. Following the ASD recipe we modeled the external behavior of the PDU first (the IPDU interface model). Then, we separately described the external behavior of the PCs located at the subsequent level of the PDU. After that, we modeled the PDU design such that it refines the IPDU specification and includes all interactions with the PCs. Below we individually introduce these models in more depth.

The external behavior of the PDU (the IPDU interface model). The specification in Figure 6 describes the external behavior with respect to the external users of the PDU, according to the original state machine of Figure 2. The specification is described using an ASD interface model. The specification is straightforward and self-explainable.

The internal events of the interface model of the PDU represent the detailed activities that may internally occur in the implemented system, not visible to the external world. For example, rule case 26 indicates that when the system is operational, something internal might happen in the system that leads the PDU to transit to the *System_Off* state. The detailed internal behavior that matches this internal event in the design of the PDU is that the ControlPC may send the *controlPowerOff* callback to the PDU and then the PDU will process this callback by sending the shutdown message and powering off all PCs.

During the refinement check using FDR established by the ModelBuilder, all events not visible to the client component will be hidden from the interface model: the *ICR_PC_INT.powerOff* and the *IGeo_PC_INT.geoStop* events, for instance. To reflect the internal modes of the system on the external specification one needs to add visible callbacks to the interface. For example, we can indicate to the user that the system is powered off in the *System_On* state due to internal activities by replacing the *Null* response of rule case 26 by an extra callback (say *IUserIndicationCB.systemOff*). This way the deep internal modes of the system can be reflected on the external specification, making the specification more transparent. We omit such extensions from our specification since we are interested more in verifying the correctness of the state machine of the internal PDU design.

The external behavior of PCs. The behavior of PCs was separately described using ASD interface models, matching the state machines introduced earlier in Figure 3. The specification of the GeoPC is introduced earlier in Figure 4. Similarly, the specification of the other PCs is straightforward, and we refer

	Channel	Stimulus event	Predicate	Response	State update	Next state	Comment	Tag
1	PDU_Off<>							
2	IPDU	PDUswitchOn		IPDU.NullRet		System_StandBy		
3	IPDU	PDUswitchOff		IPDU.NullRet		PDU_Off		
4	IPDU	powerOn		IPDU.NullRet		PDU_Off		
5	IPDU	powerOff		IPDU.NullRet		PDU_Off		
6	IPDU	forcedPowerOff		IPDU.NullRet		PDU_Off		
7	IPDU	emergencyOff		IPDU.NullRet		PDU_Off		
8	ICR_PC_INT	powerOff		Blocked		+		
9	IGeo_PC_INT	geoStop		Blocked		+		
10	System_StandBy<IPDU.PDUswitchOn>							
11	IPDU	PDUswitchOn		IPDU.NullRet		System_StandBy		
12	IPDU	PDUswitchOff		IPDU.NullRet		PDU_Off		
13	IPDU	powerOn		IPDU.NullRet		System_On		
14	IPDU	powerOff		IPDU.NullRet		System_StandBy		
15	IPDU	forcedPowerOff		IPDU.NullRet		System_StandBy		
16	IPDU	emergencyOff		IPDU.NullRet		Emergency_Off		
17	ICR_PC_INT	powerOff		Blocked		+		
18	IGeo_PC_INT	geoStop		Blocked		+		
19	System_On<IPDU.PDUswitchOn, IPDU.powerOn>							
20	IPDU	PDUswitchOn		IPDU.NullRet		System_On		
21	IPDU	PDUswitchOff		IPDU.NullRet		PDU_Off		
22	IPDU	powerOn		IPDU.NullRet		System_On		
23	IPDU	powerOff		IPDU.NullRet		System_StandBy		
24	IPDU	forcedPowerOff		IPDU.NullRet		System_Off		
25	IPDU	emergencyOff		IPDU.NullRet		Emergency_Off		
26	ICR_PC_INT	powerOff		Null		System_Off		
27	IGeo_PC_INT	geoStop		Null		Geo_Stop		
28	Emergency_Off<IPDU.PDUswitchOn, IPDU.emergencyOff>							
29	IPDU	PDUswitchOn		IPDU.NullRet		Emergency_Off		
30	IPDU	PDUswitchOff		IPDU.NullRet		PDU_Off		
31	IPDU	powerOn		IPDU.NullRet		System_On		
32	IPDU	powerOff		IPDU.NullRet		Emergency_Off		
33	IPDU	forcedPowerOff		IPDU.NullRet		Emergency_Off		
34	IPDU	emergencyOff		IPDU.NullRet		Emergency_Off		
35	ICR_PC_INT	powerOff		Blocked		+		
36	IGeo_PC_INT	geoStop		Blocked		+		
37	System_Off<IPDU.PDUswitchOn, IPDU.powerOn, IPDU.forcedPowerOff>							
38	IPDU	PDUswitchOn		IPDU.NullRet		System_Off		
39	IPDU	PDUswitchOff		IPDU.NullRet		PDU_Off		
40	IPDU	powerOn		IPDU.NullRet		System_On		
41	IPDU	powerOff		IPDU.NullRet		System_Off		
42	IPDU	forcedPowerOff		IPDU.NullRet		System_Off		
43	IPDU	emergencyOff		IPDU.NullRet		Emergency_Off		
44	ICR_PC_INT	powerOff		Blocked		+		
45	IGeo_PC_INT	geoStop		Blocked		+		
46	Geo_Stop<IPDU.PDUswitchOn, IPDU.powerOn, IGeo_PC_INT.geoStop>							
47	IPDU	PDUswitchOn		IPDU.NullRet		Geo_Stop		
48	IPDU	PDUswitchOff		IPDU.NullRet		PDU_Off		
49	IPDU	powerOn		IPDU.NullRet		System_On		
50	IPDU	powerOff		IPDU.NullRet		Geo_Stop		
51	IPDU	forcedPowerOff		IPDU.NullRet		Geo_Stop		
52	IPDU	emergencyOff		IPDU.NullRet		Emergency_Off		
53	ICR_PC_INT	powerOff		Blocked		+		
54	IGeo_PC_INT	geoStop		Blocked		+		

Fig. 6. The external behavior of the PDU towards the external users

to [6] for their detailed specification.

All PCs receive a number of messages synchronously via a number of channels: *GeoPC*, *IGeoPC_Broadcast* in the GeoPC interface, for instance. The GeoPC interface includes one internal event that models the internal behavior of pressing a Stop button on a movable part. The internal event is 'yoked' in rule case 14 which means that sending the *IGeoPC_CB.stop* callback to the queue of the PDU is restricted to a single callback at a time. In fact, increasing the yoking threshold restriction to more than one allowed callback will gives the same verification results using model checking but with more generated states, transitions and time.

Modeling the behavior of the PDU controller. The specification of the PDU controller was described by an ASD design model, matching the original state machine introduced in Figure 2. Using the ASD ModelBuilder, we included the used interface models of the PCs and explicitly specified the number of instances of each interface model before describing the behavior of the PDU design model. Obviously, our design model includes one instance of the ControlPC interface model, one instance of the GeoPC interface model and five instances of the PC interface models.

In Figure 7 we provide the specification of *System_On* and *GeoStop* states. The complete specification of the model can be found in [6]. To give an example of the usage of instances of used interfaces in the ModelBuilder consider rule case 26 of Figure 7. The rule case specifies that when the PDU controller receives the *controlPowerOff* asynchronous event from the *CR_PC:ICR_PC_CB* interface (via its queue), it executes a list of responses one by one until completion. The *CR_PC:ICR_PC_Broadcast.shutdown* denotes sending the *shutdown* message to the ControlPC instance via the *ICR_PC_Broadcast* channel synchronously (note that all calls from client to used components are synchronous in ASD). Similarly, the synchronous response *All:IPC.powerOff* in rule case 26 denotes powering off all five PCs sequentially one by one, i.e., *PC1.IPC.powerOff*, .., *PC5.IPC.powerOff*.

One of the benefits of the ASD specification is that during the specification process of the PDU design model, design decisions had been discussed early and detailed carefully to cover those scenarios that had not been considered during requirement and design phases. This had mainly been raised because the ASD specification process forces specification completeness, by filling-in and thinking about every possible stimulus in every table.

Since the original state machine of the PDU is not complete, in the sense that not all external calls or internal callback stimuli events are depicted in every state, we initially assigned the *Illegal* response to every internal callback stimulus received from the PCs if the stimulus does not appear in a state of the original state machine.

For example, we assign *Illegal* responses to the *controlPowerOff* and the *stop* callback stimuli in all states except *System_On* (see rule cases 26 and 27 in Figure 7 compared with rule cases 53 and 54). Similarly, all external user commands not present in a state are ignored, i.e., they make a self-transition in the state.

A. Formal verification of the PDU controller

Upon the completion of all ASD models, the formal verification process using model checking was started. Figure 8 depicts a screenshot of the formal checks performed remotely by the FDR model checker using the ASD ModelBuilder.

The first and the second properties check whether the IPDU interface model is livelock and deadlock free. The third, fourth and fifth properties verify that the interfaces of the PCs are livelock free. Verifying deadlock freedom can be established for each interface model separately using the ModelBuilder.

The sixth property checks whether the combined model is a deterministic design. The purpose of this check is to prevent

19	System_On<IPDU.PDUswitchOn,IPDU.powerOn>							
20	IPDU	PDUswitchOn		IPDU.NullRet		System_On		
21	IPDU	PDUswitchOff		CR_PC:ICR_PC.powerOff; GeoPC:IGeoPC.powerOff; All:IPC.powerOff; IPDU.NullRet		PDU_Off		
22	IPDU	powerOn		CR_PC:ICR_PC_Broadcast.restart; GeoPC:IGeoPC_Broadcast.restart; All:IPC_Broadcast.restart; IPDU.NullRet		System_On		
23	IPDU	powerOff		GeoPC:IGeoPC_Broadcast.shutdown; All:IPC_Broadcast.shutdown; GeoPC:IGeoPC.powerOff; All:IPC.powerOff; IPDU.NullRet		SystemStandby	Send shutdown Start timer sw tabs off	
24	IPDU	forcedPowerOff		CR_PC:ICR_PC.powerOff; GeoPC:IGeoPC.powerOff; All:IPC.powerOff; IPDU.NullRet		System_Off	All taps off	
25	IPDU	emergencyOff		CR_PC:ICR_PC.powerOff; GeoPC:IGeoPC.powerOff; All:IPC.powerOff; IPDU.NullRet		Emergency_Off		
26	CR_PC:ICR_PC_CB	controlPowerOff		CR_PC:ICR_PC_Broadcast.shutdown; GeoPC:IGeoPC_Broadcast.shutdown; All:IPC_Broadcast.shutdown; CR_PC:ICR_PC.powerOff; GeoPC:IGeoPC.powerOff; All:IPC.powerOff		System_Off	Send shutdown Start timer All tabs off	
27	GeoPC:IGeoPC_CB	stop		Null		Geo_Stop	Movable parts off	
46	Geo_Stop<IPDU.PDUswitchOn,IPDU.powerOn,GeoPC:IGeoPC_CB.stop>							
47	IPDU	PDUswitchOn		IPDU.NullRet		Geo_Stop		
48	IPDU	PDUswitchOff		CR_PC:ICR_PC.powerOff; GeoPC:IGeoPC.powerOff; All:IPC.powerOff; IPDU.NullRet		PDU_Off		
49	IPDU	powerOn		IPDU.NullRet		System_On	Movable parts on	
50	IPDU	powerOff		IPDU.NullRet		Geo_Stop		
51	IPDU	forcedPowerOff		IPDU.NullRet		Geo_Stop		
52	IPDU	emergencyOff		CR_PC:ICR_PC.powerOff; GeoPC:IGeoPC.powerOff; All:IPC.powerOff; IPDU.NullRet		Emergency_Off		
53	CR_PC:ICR_PC_CB	controlPowerOff		Illegal		-		
54	GeoPC:IGeoPC_CB	stop		Illegal		-		

Fig. 7. The specification of *System_On* and *GeoStop* states

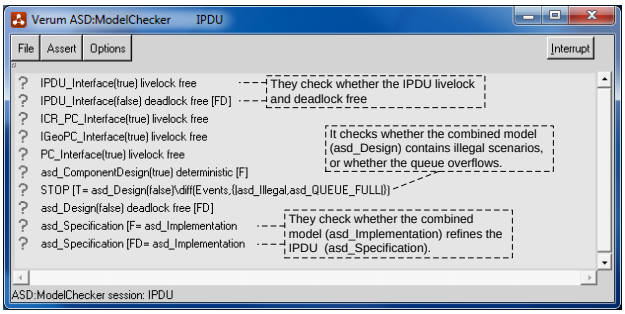

Fig. 8. Formal checks performed by FDR for the behavioral verification

ambiguities in the generated source code when compiled with the rest of the product code.

The seventh property searches for events marked as illegal and queue overflow scenarios in the combined model (asd_Design). The eighth property verifies that the combined model is deadlock free. While the last two properties are used to check whether the combined model (asd_Implementation) refines the IPDU interface model (asd_Specification), under both the Failure and Failure-Divergence models.

We performed the behavioral verification of the PDU step by step. We first began by checking the existence of illegal scenarios in the combined model of the PDU (the seventh check). The FDR model checker detected a major error embedded in the design of the PDU. The FDR counterexample is visualized in the sequence diagram of Figure 9.

The practical scenario of the potential consequences of this error is as follows. During the regular execution of the system (the PDU is in the *System_On* state), the clinical users may experience some issues related the the movable parts. Consequently, the clinical users might choose to press the *Stop* buttons attached to the body of the movable parts. After the GeoPC has sent the *Stop* signal, the PDU immediately switches off the power that supplies the movable parts, transits to the *Geo_Stop* state, and waits for the subsequent user input. But the user might desire to entirely power off the system for safety purposes. This appeared to be impossible in the current design.

More precisely the user of the ControlPC would not be able to power off the system via the *controlPowerOff* signal when the PDU is in the *Geo_Stop* state, and also both *powerOff* and *forcePowerOff* commands would have no effect on the PDU. Furthermore, if the user chooses to strictly cut the power down from the mains switch, the UPS would start automatically, if it is attached, and the erroneous situation would remain. Only pressing the *Emergency* button would rescue the user from this case since it entirely cuts down the power to the system.

The benefits of specification completeness plus the formal verification using model checking for detecting the error is obvious here. Assigning the *Illegal* response to the absent callback stimuli in the original state machine had effectively helped us detecting the veiled error. Without specification completeness such error is hard to detect. For example, if we assign *Null* instead of *Illegal* responses, the entire model becomes deadlock and Illegal free. Therefore, to detect the above error one needs an explicit requirement to be formulated

as a formal property and checked against the entire model. But, indeed one might not consider the exact property that detects the error if the corresponding requirement is missing or if the error is not known a priori.

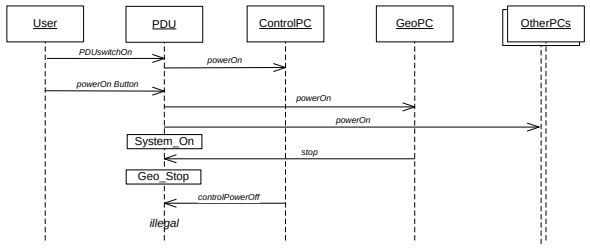

Fig. 9. FDR counterexample

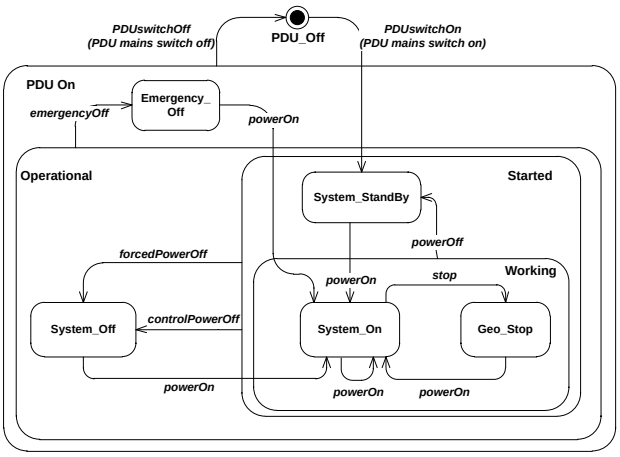

Fig. 10. The improved state machine after formal verification and specification review

Another design error was found during the specification review of the models by team members, due to the evolution of requirements. Consider the state machine of Figure 2 once more. The *forcedPowerOff* and the *controlPowerOff* transitions from the *System_Standby* state to *System_Off* state were found missing. This means that the clinical user would not be able to power off the ControlPC upon pressing the *PowerOff* button for 10 seconds or systematically power off the system using the *controlPowerOff* signal when the system is in the *System_Standby* state. Initially, this was a desired behavior since the ControlPC should always be operational, but lately a decision was made to also consider powering off the ControlPC in the *System_Standby* state.

B. The improved PDU controller

After the design errors had been communicated to the PDU designers, the design had been adapted. The modified state machine of the PDU is depicted in Figure 10. It includes the missing *forcedPowerOff* and *controlPowerOff* transitions from *System_Standby* state to *System_Off* state. Additionally, the modified state machine allows the clinical users to power off the system when the PDU is in the *Geo_Stop* state.

Subsequently, the specification of the PDU design model had been adapted to the changes. All responses to internal callback stimuli received from PCs not specified in the original state machine were set to *Null*. Moreover, the ASD specification of the *System_Standby* and the *Geo_Stop* states had been adapted. The complete specification of the improved PDU design model and the corresponding external specification can be found in [6].

All properties listed in Figure 8 succeeded except the last property. For this property FDR reports six counterexamples in total where divergences might occur, affecting the external behavior. The analysis of these counterexamples reveals that the source of all divergence scenarios is basically the same. The sequence diagram that explains the erroneous scenario is visualized in Figure 11.

The counterexample shows that the GeoPC could continuously inform the PDU about a pressed *Stop* button on the body of a movable part. Then, the PDU endlessly treats the *stop* signals, with the possibility that the external user commands are not treated immediately.

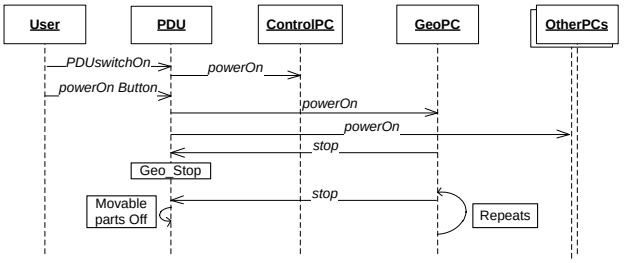

Fig. 11. A divergence that affects the external behavior of the PDU

We don't really consider these divergences as critical errors. They are rather benign, but they can happen indeed. Furthermore, in the real system there is still a possibility to power off the entire system using the emergency or the power off button, despite the existence of a desperate user hitting the stop-button.

V. MODELING EFFORTS

The activities of modeling and verifying the behavior of the PDU were conducted part time in parallel with other traditional activities devoted to the PDU development. Understanding the PDU domain plus studying various design documents [10], [9], [11] for the purpose of modeling the behavior of the PDU took approximately 35 hours.

The modeling and verification efforts of all ASD models took 32 hours in total. In general, the effort of creating the ASD models was not a time demanding process, because of the high-level description provided by the ASD ModelBuilder. The team involved in modeling, specification, verification and review processes were highly skilled in traditional development methods, but had limited knowledge of formal methods.

But despite this limitation, team members were able to quickly understand and review the ASD specification, and to favorably provide their feedbacks and suggestions for improvements although no one of the reviewers had previously been exposed to any ASD training courses.

Since development team had limited knowledge of formal mathematical methods, especially model checking technologies, the ASD technology seems to be suitable and acceptable in the industry domain.

Table II depicts the statistical data of the ASD models. The first column lists the name of all ASD interface and design models, related to the PDU and the PCs. The second column contains the total number of rule cases, specified and reviewed by team members. The third, fourth, and fifth columns include statistical data produced by the FDR model checker for checking deadlocks of each model independently. All models are deadlock free. Note that the data presented for the PDU (design) model is related to the combined model that includes the parallel composition of the PDU design plus the interface models of the PCs. The third column depicts the number of generated states, while the fourth column presents the number of generated transitions. The time required for verifying each model was less than one second.

Model	Rule cases	States	Transitions
IPDU	54	16	67
PDU	54	824	1,360
ICRPC	18	16	25
IGeoPC	18	16	25
OtherPCs	15	15	23

TABLE II
STATISTICAL DATA RELATED TO THE ASD MODELS OF THE PDU

The PDU team decided to continue the development of the PDU controller using the ASD technology and to further investigate other design alternatives. The development process of the PDU was continued by other team members newly introduced to the ASD method. The team investigated further other design alternatives, and applied the technology to the development of various parts of the X-ray system, especially of the services deployed on the PCs that communicate with the PDU.

Finally, the formal behavioral verification, team reviews and the specification completeness processes performed throughout this work provided a proper framework for improving the real PDU, for assisting the development of the controller, and decreasing potential effort devoted to error fixing at later stages of the project.

Future work. As a logical following step of modeling the PDU design, the behavior of the PDU and the PCs will be extended such that they include all intermediate states. Error monitoring and recovery operations will also be considered. We also plan to extend the external specification and the design of the PDU with extra callback events that reflect the internal states of the system: callbacks indicating that the system is on, off, starting up or in *Geo_Stop* state, for instance.

The behavior of the forthcoming model is foreseen to be rather complex due to interleaving caused by the concurrent PCs during the transiting states, so that avoiding the state space explosion problem is a challenging task. We believe that systems can to be designed such that they become easily verified [5], so that the number of generated states depends not only on the complexity of systems but also on the way they have been designed.

Therefore, we are planning to gradually extend the current model of the PDU and to carefully investigate a number of specification techniques to avoid the state space explosion problem from [5]. We compare a number of design and specification styles, for example between a design of the PDU that uses a pushing strategy (where PCs notify the PDU about their states) and another alternative design that employs a polling mechanism (where the PDU queries the states of the PCs when needed).

We are further planning to model the extended design of the PDU directly using CSP/FDR or mCRL2 [12] to exploit some of their useful features. The reason is that we will have more flexibility to formulate formal properties, apply manual abstractions, and to further exploit some compression techniques such as the hierarchal compression of state spaces [4] offered by the FDR model checker or the confluence reduction and branching-bisimulation reduction [12] supplied by the model checker mCRL2, in case the state space explosion problem is about to happen.

REFERENCES

[1] G.H. Broadfoot. ASD case notes: Costs and benefits of applying formal methods to industrial control software. In *FM 2005: Formal Methods*, volume 3582 of LNCS, pages 548–551. Springer (2005), 2005.

[2] J.M. Carter and J.H. Poore. Sequence-based specification of feedback control systems in Simulink®. In *CASCON '07: Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, pages 332–345, New York, NY, USA, 2007. ACM.

[3] I. Crnkovic. *Building Reliable Component-Based Software Systems*. Artech House, Inc., Norwood, MA, USA, 2002.

[4] FDR homepage. *http://www.fsel.com*.

[5] J.F. Groote, T.W.D.M. Kouters, and A.A.H. Osaiweran. Specifcation guidelines to avoid the state space explosion problem. In *Proceedings of the 4th IPM international Conference, FSEN 2011*, page (IN PRESS), Tehran, Iran, 2011. Springer-Verlag, Berlin, Germany.

[6] J.F. Groote, A. Osaiweran, and J.H. Wesselius. Analyzing a controller of a power distribution unit using formal methods. CS-Report 11-14, Eindhoven University of Technology, 2011.

[7] J.F. Groote, A. Osaiweran, and J.H. Wesselius. Experience report on developing the frontend client unit under the control of formal methods. In *Proceedings of the 27th ACM Symposium on Applied Computing*. ACM Press, IN PRESS, 2012.

[8] J.F. Groote, A. Osaiweran, and J.H. Wesselius. Analyzing the effects of formal methods on the development of industrial control software. In *proceedings of the 27th IEEE ICSM 2011*, pages 467–472, Williamsburg, VA, USA, September 25-30, 2011.

[9] R. Kleihorst. Feature design - foundation power distribution subsystem, internal Philips document. 2010.

[10] R. Kleihorst. Power distribution unit - concept specification, internal Philips document, xdy036-080190. 2010.

[11] M. Loos. Feature design - startup/shutdown, internal Philips document, v0.6. 2010.

[12] mCRL2 toolset homepage. *http://www.mcrl2.org/*.

[13] A.W. Roscoe. *The theory and practice of concurrency*. Prentice Hall, 1998.

[14] Verum homepage. *http://www.verum.com*.