Modelling and Analysis of a Redundant Mobile Robot Architecture Using AADL

Geoffrey Biggs[1], Kiyoshi Fujiwara[1], and Keiju Anada[2]

[1] Intelligent Systems Research Institute
National Institute of Advanced Industrial Science and Technology (AIST)
AIST Tsukuba Central 2, Tsukuba, Ibaraki 305-8568, Japan
[2] aRbot, Inc., Japan

Abstract. As the complexity of robots deployed in the real world increases, the use of formal specifications in the development of safety-critical robot systems is becoming increasingly important. A formal specification gives confidence in the correctness, completeness, and accuracy of a system design. In this paper, we present a formal specification of a redundant control architecture for a mobile robot in the form of a model. The model is created using the Architecture Analysis and Design Language (AADL). This formal language allows the model to be analysed to prove system properties of interest. In this case, we are interested in proving the response time of the robot to external obstacles and to internal errors. We present the model and the results of these analyses with the goal of proving that the architecture is sufficiently safe for use in a safe robot wheelchair.

1 Introduction

As with any cyber-physical or embedded system, an important part of designing a robot is specifying the architecture of the control system that turns sensor readings and planned actions into controlled motions of the robot. Sensors, micro-processors, control software, actuators, and the communication buses that connect them must all be designed to provide sufficient capacity, in each respective way, to perform their role in the overall control system.

For system correctness, it is not enough to simply design the control system. It is also necessary to ensure that the design will satisfy the system's requirements. When the system being developed is safety-critical, there is a further need to provide proof that a design is correct and satisfies the system's functional and non-functional requirements.

An approach to both ensuring and proving that a design is suitable is the application of formal methods, including formal specifications of design. The use of formal methods during the design of cyber-physical and embedded systems is becoming increasingly common. There have been several noteworthy projects to produce complete tool chains using formal methods for system design, such as the TOPCASED project [9]. Their continued growth is an indication of the

D. Brugali et al. (Eds.): SIMPAR 2014, LNAI 8810, pp. 146–157, 2014.

success they have had in introducing formally-based methodology to fields such as aerospace and railway development.

Formal methods allow a design to be proven to be correct, in that it contains no mistakes in specification. For example, designed wiring between components can be shown to be correct in such terms as the information carried, or that all necessary connections exist in the design. Formal specifications of behaviour, for example finite automata, can ensure that the system's behaviour is defined for all known inputs. The use of formal methods can reduce the time to develop a system through reduction of other parts of the development process; for example, the IEC 61508 standard for Functional Safety of Programmable Electrical/Electronic Devices specifically states that the use of formal methods in design of software reduces the necessary testing that must be performed [2,3].[1] Analyses performed using a formal method can even aid in debugging problems found during implementation, such as solving a bottleneck in system response by identifying parts of the system that may experience high latencies.

This article describes the application of a formal method during the design of the control system architecture for a safety-monitoring motorised wheelchair. The wheelchair applies robot control technology to avoid collisions when possible and reduce impact forces when not. It also ensures that the wheelchair is robust to failures and is guaranteed to come to a safe stop in the case of one. A formal specification language was used to specify the control architecture of the wheelchair. Formal analyses were performed on this specification to determine such factors as the latency in responding to the appearance of an obstacle, the response time to a failure, and the suitability of the chosen micro-controller hardware. Based on the analyses, we claim that the control system architecture is correct and sufficient to provide reasonable safety to a user of a motorised wheelchair under the analysed scenarios.

The next section discusses the formal specification language used in this work. Following that, the formal model of the control system architecture is given in Section 3. Analysis results are given in Section 4.

2 AADL

The Architecture Analysis and Design Language (AADL) is a modelling language for the formal modelling of embedded system architectures [4]. Its goal is modelling and formally proving the correctness of cyber-physical and embedded system architectures, allowing the architecture of even extremely complex systems to be designed accurately It is a declarative language, focusing on structure rather than behaviour. It supports modelling a system's software structure, the structure of the execution hardware, the mapping of software to execution hardware, and the sensors, actuators and similar devices that provide interfaces to the external world.

[1] The standard *requires* the use of formal methods at higher safety integrity levels – see Table A.2 of Part 3 of the standard.

AADL has a formal semantics and fixed syntax. This means that it is possible to check a model for correctness. The capacity to specify properties of all components in the model, including software, processors, and communication buses, allows a developer to perform model analysis to examine properties of the system such as execution time or communication bandwidth utilisation. We describe some analyses in Section 4.

AADL was chosen for this work due to its formality. We have previously used SysML to model the entire wheelchair system, including its control architecture, during the design stages [5]. SysML added structure to the design information that was particularly a benefit in establishing traceability between system requirements and system design; we were able to establish that each functional requirement relating to necessary features of the system was satisfied. However, we found that the semi-formal nature of the SysML model limited its usefulness both for proving design correctness and for analysing properties of the control architecture. By contrast, AADL's formality makes such analyses simple. At the same time, AADL is tightly focused on the embedded control system's structural design, whereas SysML can model a wider range of information such as requirements, testing and behaviour.[2]

There are several tools available for specifying and analysing AADL models. In this work, we use the Open Source AADL Tool Environment (OSATE) tool version 2.0.5 [7]. It provides a compiler for AADL models, syntax and semantic error checkers, a model correctness checker, and several analysis tools including connection analysis, thread execution schedule analysis, and flow latency analysis.

The next section describes our application of AADL to the design of the safety-monitoring wheelchair's control architecture. Our use of the OSATE tool to perform analyses on this model is described in Section 4.

3 Safe Mobile Robot Architecture Model

A model of the control architecture for a safety-monitoring wheelchair, described in [8], has been developed using AADL. This architecture is designed to conform to SIL2 of the IEC 61508 standard for electrical/electronic/programmable electronic safety-related systems [1]. This section briefly summarises the key points of the architecture and describes the model.

3.1 Architecture Description

The control architecture modelled in this work is used to control the wheel speed of a motorised wheelchair. Wheel speed must be controlled in response to velocity command inputs from an external system, such as a human operating a joystick.

The controller provides two safety aspects. The first aspect is safety against collisions. Range sensors cover the wheelchair's entire surroundings, watching for

[2] Planned and recently-produced extensions to the AADL standard (known as "annexes") add support for error modelling and requirements modelling.

obstacles. The wheelchair's velocity is limited appropriately when obstacles are near. Additionally, a contact sensor detects impacts on the wheelchair, bringing it to an immediate halt when one is detected.

The second aspect is safety against failures. Redundancy is used in the form of dual CPUs. They each control a single motor/wheel and run identical software (with an exception for configuring one to rotate its motor in the opposite direction). The two CPUs also continuously monitor each other. If a failure is detected, such as non-responsiveness or an incorrect response to a command or sensor input, the partner CPU will halt its own motor.[3]

The redundant controller architecture has been implemented on real hardware. The execution hardware used is two Renesas SH72AW CPUs, one for each redundant motor control unit. These run at 160 MHz and have 96 kB of RAM. Two AC motors are used, one for each wheel/drive unit. Each motor controller/-motor combination drives a single wheel. Four Hokuyo UAM-01LP-T301 laser range finders, which are certified to SIL2 of IEC 61508, are used for ranged sensing, and a fault-tolerant strip switch provides contact sensing.

3.2 AADL Model

This section presents an AADL model of the control architecture described in Section 3.1. The model describes only the structural architecture of the wheelchair control system.

The model is organised into several systems and black box devices that are composed to produce the whole control system. This top-level structure is shown in Figures 1 (information connections) and 2 (hardware buses). The laser range finders, touch sensors, and the motors and their related control hardware are all treated as black boxes with known external interfaces and known activity periods.[4] Note also the presence in the model of several buses, such as the wire connections to the sensors and the CAN bus connections to the motors. The presence of execution hardware such as this in the model allows for analysis of sense-compute-actuate latencies and bus bandwidth utilisation. In this model, each individual bus has been specified to allow the analysis of bus bandwidth use.

The two redundant CPUs and the software they execute (named a "control unit" in the discussion below) are each represented by instances of the same subsystem design. This sub-system is shown in Figure 3. Each control unit contains both the software to be executed (the ctrl_proc process) and the computing hardware to execute it. The software itself is divided into five separate tasks, shown in Figure 4. These each have a role in turning sensor and control inputs into motor commands while monitoring safety. Each thread's required real-time execution deadline and processor budget is specified in the model. Modelling the software allows for analysis of processor load and deadline achievement.

[3] Halting the motor of the failed CPU is achieved through a watchdog timer on the motor control input of the motor controller and brakes that engage when power is removed.

[4] The activity periods of sensors and actuators are necessary during latency analyses.

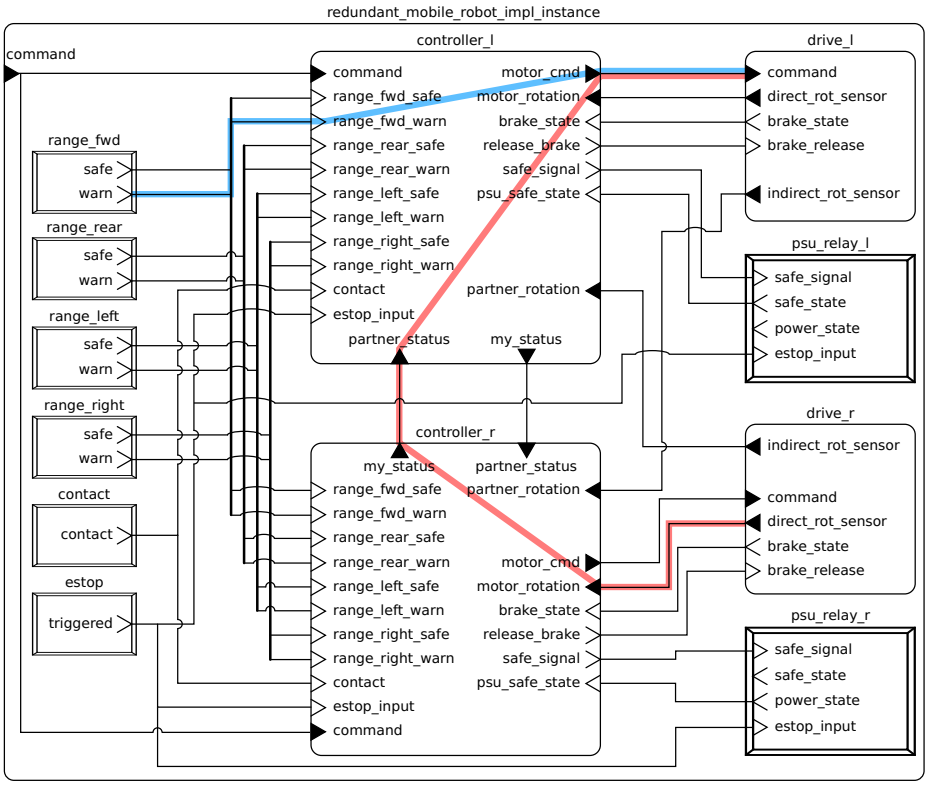

Fig. 1. The information connections at the top level of the control architecture model

The drive units are similarly represented by two instances of the same subsystem design, shown in Figure 5. Each drive unit includes an AC motor (which includes a rotation sensor), a brake, a separate rotation sensor using a different technology to that used in the in-built sensor of the motor, and the motor control unit that turns motor torque commands into AC voltages to drive the motor.

The model contains 796 SLOC of AADL.

4 Formal Analysis

The formal semantics of AADL allow a variety of analyses to be performed on an instantiated model. The limit to what can be analysed is determined by the available tools and the properties provided in the model. This section describes some of the analyses performed on the safety architecture model.

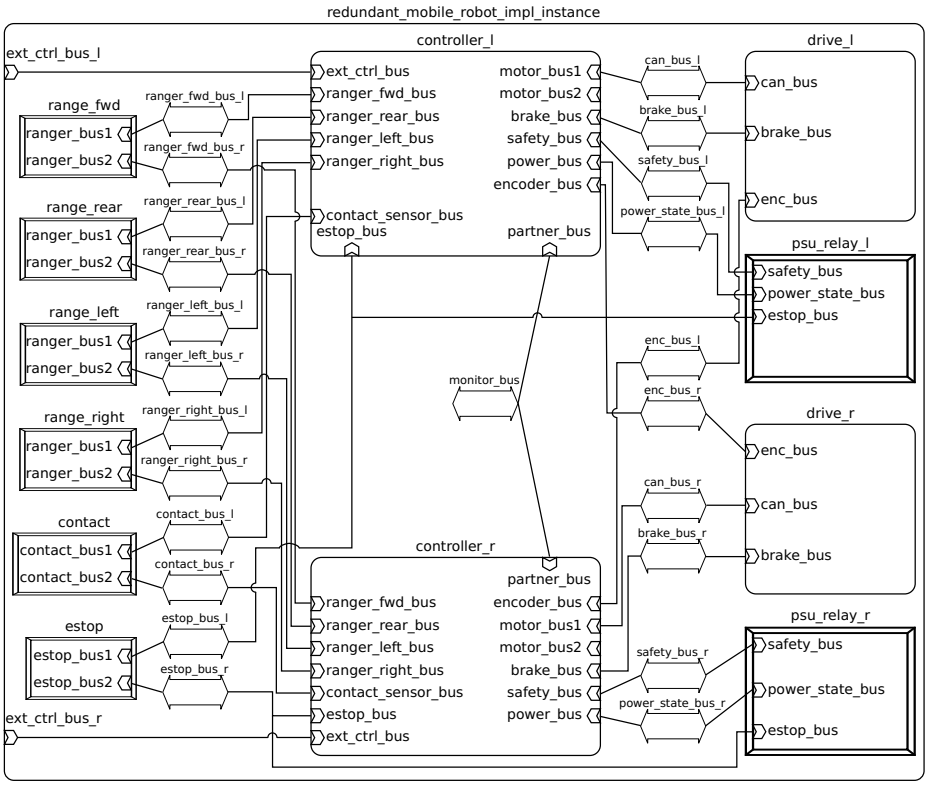

Fig. 2. The buses at the top level of the control architecture model

4.1 Obstacle Response Latency

The response time of an embedded system depends on the latency between receiving an input and producing an output. In AADL, such latencies can be modelled and calculated using *flows* [6]. A flow is a path through a system specified by the model developer. Typically it is from an input (the *source*), through the communication and processing of that input (the *path*), and to an output (the *sink*). Flows may be hierarchically specified, with one flow making use of sub-flows along its path.

We have used flows to analyse two latencies in the control system that are important to safety. The first, described below, is the latency of responding to the detection of an obstacle. The second, described in the next section, is the latency of one drive unit responding to an error in the opposite side's drive unit.

The flow of signals through the control system from a range sensor to the motor controlling a wheel is illustrated in Figures 1 to 5, highlighted in blue.

The results of the latency calculations performed by OSATE for this flow are given in Table 1. The latency is calculated for the worst case. It includes the

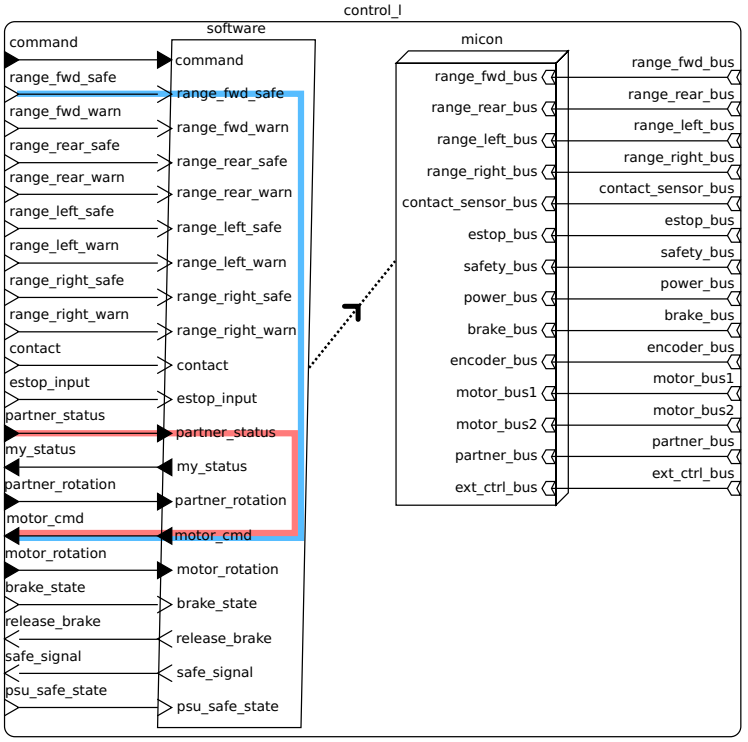

Fig. 3. The structure of one of the redundant control units, including software and execution hardware

maximum possible delay from the range sensor (30 ms, corresponding to its cycle time), delays due to sampling and scheduling of software in the processor, the communication delay across the CAN bus to the motor controller (at a transfer speed of 1 $Mbps$), and the response time of the motor controller and motor itself, excluding physically-limited response time.

The calculated total worst-case latency is 160.064 ms. At the maximum velocity of the robot of 8 km/h, it corresponds to a distance of 0.356 m. At the more common velocity of 4 km/h, it is 0.178 m.

4.2 Partner Monitoring Latency

The worst-case latency of responding to an error has been calculated. The error used is the left drive unit responding to a failure in the right drive unit, manifesting as a mis-match between the right encoder reading and what the right control unit reports for motor rotation. The flow of signals through the control system from the right rotation sensor to the motor controlling the left wheel is illustrated in Figures 1 to 5, highlighted in red.

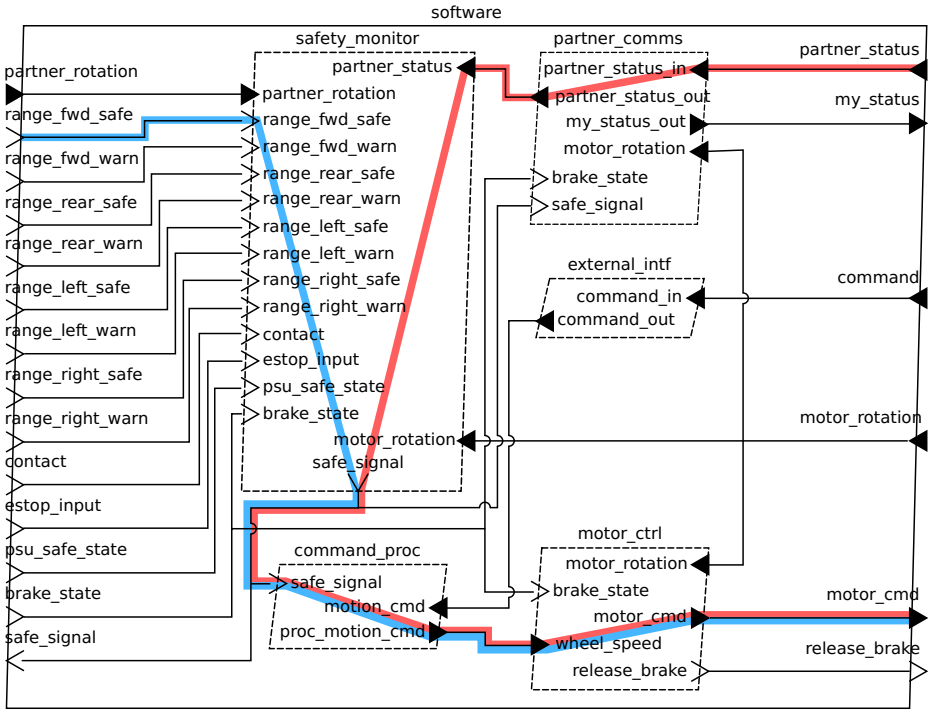

Fig. 4. The structure of the software that executes within each microprocessor

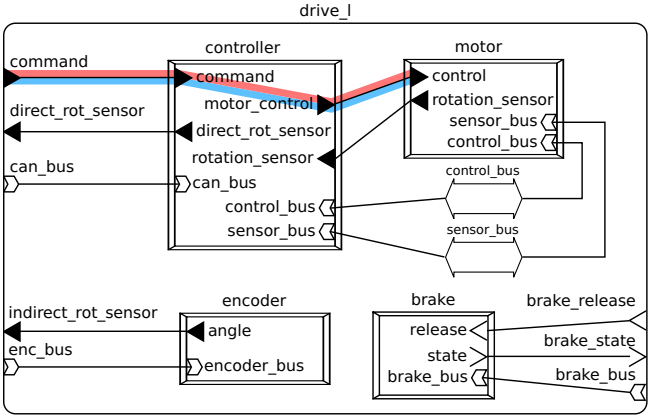

Fig. 5. The structure of each drive unit

Table 1. The latency of responding to an obstacle, as calculated by OSATE. Time columns are the real-time deadline, time due to sampling delay, time due to communication, and additional time.

Model element	Name	Deadline or Connection	Sampling	Flow spec	Additional	Total
device	range_fwd	0.0 μs	0.0 μs	30.0 ms	30.0 ms	30.0 ms
connection	range_fwd.warn → control_l.software.safety_monitor.range_fwd_warn	0.0 μs	0.0 μs	0.0 μs	30.0 ms	30.0 ms
thread	control_l.software.safety_monitor:obs_flow	10.0 ms	10.0 ms	0.0 μs	20.0 ms	40.0 ms
connection	safety_monitor.safe_signal → command_proc.safe_signal	0.0 μs	0.0 μs	0.0 μs	20.0 ms	60.0 ms
thread	control_l.software.command_proc:unsafe_flow	10.0 ms	10.0 ms	0.0 μs	20.0 ms	70.0 ms
connection	command_proc.proc_motion_cmd → motor_ctrl.wheel_speed	0.0 μs	0.0 μs	0.0 μs	20.0 ms	90.0 ms
thread	control_l.software.motor_ctrl:cmd_flow	10.0 ms	10.0 ms	0.0 μs	10.0 ms	100.0 ms
connection	control_l.software.motor_ctrl.motor_cmd → drive_l.controller.command	64.0 μs	0.0 μs	0.0 μs	10.064 ms	
device	drive_l.controller:cmd_flow	50.0 ms	0.0 μs	50.0 ms	60.064 ms	100.0 ms
connection	controller.motor_control → motor.control	0.0 μs	0.0 μs	0.0 μs	60.064 ms	160.064 ms
device	drive_l.motor:cmd_flow_sink	0.0 μs	0.0 μs	0.0 μs	60.064 ms	100.0 ms
Total						160.064 ms

Unfortunately, a bug in the version of OSATE used prevented us from calculating the latency of the complete path. An exception is triggered when calculating the latency of the serial link between the two controllers, preventing the latency calculation from completing. To work around this, we split the flow into two, one for each side of the partner monitoring link.

The latency of the flow from its source to the my_status port of controller_r is given in Table 2. This is the time for the right control unit to produce a status message. The latency of the flow from the partner_status port of controller_l to its sink is given in Table 3. This is the time for the left control unit to detect and respond to the error. As in the previous analysis, it includes the various delays and latencies involved in controlling the motor.

The latency of the serial monitoring link between the two controllers has been calculated as 2.08 ms. This is based on a transmission speed of 38400 bps and a message size (from the model) of 10 bytes.

Based on the two latency values and the manual calculation of the latency of the serial monitoring link, the total worst-case latency is 142.186 ms. At the maximum velocity of the robot of 8 km/h, it corresponds to a distance of 0.316 m. At the more common velocity of 4 $km.h$, it is 0.158 m.

5 Discussion

The latency analyses of the previous section indicate that the control architecture provides safety in the two analysed scenarios. However, we say "indicate" here rather than "prove" due to the problem with the tool mentioned in Section 4.2. The failure in calculating one of the latencies led us to doubt the calculation results, and a manual estimate had to be calculated to confirm they are in the expected range, but even then we do not consider them to be absolute proof.

A formal specification gives confidence in the correctness of the specification. It does so through correctness checks on the specification for internal consistency and consistency with the formal model in use. In our case, these checks are performed when the AADL model is compiled; they confirm that the specification is error-free, but not that the design is error-free.

Analyses on the specification give confidence in the design, and are used to confirm that the design is error-free with regard to the requirements. In the analyses presented in this paper, this is the need for the control architecture to have a sufficiently fast response time.

In both cases, however, we are relying on a tool. Our experience illustrates the importance of having reliable tools; without a tool that can be trusted to be correct, a formal specification drops in value. Manually checking for errors and performing analyses is far less reliable than automated checks and analyses, and automation is essential for large systems.

Fortunately, AADL is a standardised language. There are several for-cost tools in existence that may be more reliable than the free OSATE tool. Our experience shows the need to choose tools carefully, but it does not negate the benefits of AADL.

Table 2. The latency of responding to an error (right controller side), as calculated by OSATE

Model element	Name	Deadline or Connection	Sampling	Flow spec	Additional	Total
device	drive_r.motor	0.0 μs	0.0 μs	0.001 μs	0.001 μs	0.001 μs
Connection	motor.rotation_sensor → controller.rotation_sensor	0.0 μs	0.0 μs	0.0 μs	0.001 μs	0.001 μs
device	drive_r.controller:mot_rot_flow	0.1 μs	0.0 μs	0.1 μs	0.101 μs	0.101 μs
Connection	drive_r.controller.direct_rot_sensor → control_r.software.partner_comms.motor_rotation	32 μs	0.0 μs	0.0 μs	32.101 μs	32.101 μs
thread	control_r.software.partner_comms:motrot_out_flow	10.0 ms	10.0 ms	0.0 μs	10.0 ms	10.032101 ms
Total						20.032101 ms

Table 3. The latency of responding to an error (left controller side), as calculated by OSATE

Model element	Name	Deadline or Connection	Sampling	Flow spec	Additional	Total
thread	control_l.software.partner_comms	0.0 μs	10.0 ms	0.0 μs	10.0 μs	10.0 μs
Connection	partner_comms.partner_status_out → safety_monitor.partner_status	0.0 μs	0.0 μs	0.0 μs	10.0 μs	10.0 μs
thread	control_l.software.safety_monitor:error_flow	10.0 ms	10.0 ms	0.0 μs	10.0 ms	10.01 ms
Connection	safety_monitor.safe_signal → command_proc.safe_signal	0.0 μs	0.0 μs	0.0 μs	10.0 ms	20.01 ms
thread	control_l.software.command_proc:unsafe_flow	10.0 ms	10.0 ms	0.0 μs	20.0 ms	30.01 ms
Connection	command_proc.proc_motion_cmd → motor_ctrl.wheel_speed	0.0 μs	0.0 μs	0.0 μs	20.0 ms	50.01 ms
thread	control_l.software.motor_ctrl:cmd_flow	10.0 ms	10.0 ms	0.0 μs	10.0 ms	60.01 ms
Connection	control_l.software.motor_ctrl.motor_cmd → drive_l.controller.command	64.0 μs	0.0 μs	0.0 μs	10.064 μs	70.074 ms
device	drive_l.controller:cmd_flow	50.0 ms	0.0 μs	50.0 ms	60.064 ms	60.01 ms
Connection	controller.motor_control → motor.control	0.0 μs	0.0 μs	0.0 μs	60.064 ms	120.074 ms
device	drive_l.motor:cmd_flow_sink	0.0 μs	0.0 μs	1.0 μs	60.064 ms	60.01 ms
Total						120.074 ms

The information stored in the AADL model enables many other analyses relevant to the correct design of robots. For example, the specification can be checked for fairness to all entities using a communication bus, based on the rate of message production of each entity. Such analyses depend on tool support; future work should focus on producing analysis tools useful to robot systems.

6 Conclusions

In this work, we have used the AADL formal specification language to specify the design of the redundant control architecture of a safety-monitoring motorised wheelchair. The formal semantics and syntax of AADL shows that the specification is correct.

Using this model, we have formally analysed the control architecture design to indicate that the worst-case latency of response to sensor input and failure is sufficiently low. In both cases, the response time is sufficiently short given the maximum speed of the wheelchair. The dominant factor in safety response speed can be considered the braking speed of the wheelchair, not the control system response time.

References

1. Functional safety of electrical/electronic/programmable electronic safety-related systems. International Electrotechnical Commission (IEC) (2010)
2. Functional safety of electrical/electronic/programmable electronic safety-related systems Part 2: Requirements for electrical/electronic/programmable electronic safety-related systems, ch. 7, p. 40. International Electrotechnical Commission (IEC) (2010)
3. Functional safety of electrical/electronic/programmable electronic safety-related systems Part 3: Software requirements, ch. 7, pp. 35–36. International Electrotechnical Commission (IEC) (2010)
4. Architecture Analysis & Design Language (AADL) (AS5506B). SAE International (2012)
5. Biggs, G., Sakamoto, T., Fujiwara, K., Anada, K.: Experiences with model-centred design methods and tools in safe robotics. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 3915–3922 (November 2013)
6. Feiler, P., Hansson, J.: Flow latency analysis with the Architecture Analysis & Design Language (AADL). Tech. rep., Software Engineering Institute, Carnegie-Mellon University (2008)
7. Feiler, P.H., Gluch, D.P.: Model-Based Engineering with AADL, ch. 15. Addison-Wesley, Westford (2012)
8. Fujiwara, K., Nakabo, Y., Anada, K., Biggs, G., Mizuguchi, D.: The prototype hardware of the dependable robotic cart. In: Proceedings of the 2012 JSME Conference on Robotics and Mechatronics (2012)
9. Topcased, `http://www.topcased.org/`