Safety-Critical Java Level 2: Applications, Modelling, and Verification

Matthew Luckcuck

PhD

University of York

Computer Science

September 2016

Abstract

Safety-Critical Java (SCJ) introduces a new programming paradigm for applications that must be certified. To aid certification, SCJ is organised into three compliance levels, which increase in complexity from Level 0 to Level 2. The SCJ language specification (JSR 302) is an Open Group Standard, but it does not include verification techniques. Previous work has addressed verification for Level 0 and Level 1 programs. This thesis supports the much more complex SCJ Level 2 programs, which allow for the programming of highly concurrent multi-processor applications with Java threads, and wait and notify mechanisms.

The SCJ language specification is clear on what constitutes a Level 2 program but not why it should be used. The utility of Levels 0 and 1 are clear from their features. The scheduling behaviour required by a program is a primary indicator of whether or not Level 0 should be used. However, both Levels 1 and 2 use concurrency and fixed-priority scheduling, so this cannot be used as an indicator to choose between them. This thesis presents the first examination of utility of the unique features of Level 2 and presents use cases that justify the availability of these features.

This thesis presents a technique for modelling SCJ Level 2 programs using the state-rich process algebra *Circus*. The model abstracts away from resources (for example, memory) and scheduling. An SCJ Level 2 program is represented by a combination of a generic model of the SCJ API (the framework model) and an application-specific model (the application model) of that program. The framework model is reused for each modelled program, whereas the application model is generated afresh.

This is the first formal semantics of the SCJ Level 2 paradigm and it provides both top-down and bottom-up benefits. Top-down, it is an essential ingredient in the development of refinement-based reasoning techniques for SCJ Level 2 programs. These can be used to develop Level 2 programs that are correct-by-construction. Bottom-up, the technique can be used as a verification tool for Level 2 programs. This is achieved with the Failures Divergences Refinement checker version 3 (FDR3), after translating the model from *Circus* to the machine readable version of CSP (CSP_M). FDR3 allows animation and model checking, which can reveal sources of deadlock, livelock, and divergence. The CSP_M version of the model fits the same pattern, with a generic model of the API being combined with an application-specific model of the program. Because the model ignores scheduling, these checks are a worst-case analysis and can give false-negatives.

Contents

List of Figures

List of Tables

Acknowledgements

There are many people who deserve my gratitude for their help during the last four years. Firstly, I must thank my supervisors, Professors Ana Cavalcanti and Andy Wellings, for their patience, support, and guidance during my PhD. I will always be grateful for the time they both dedicated to helping me complete this thesis. I must also thank Professor Alan Burns, my internal examiner, for his feedback and support as my work progressed. I am also grateful to Dr Helen Treharne for taking the time to examine my thesis and provide such detailed feedback.

I would like to thank the EPSRC for the funding to undertake my PhD. I would also like to thank the staff of the Department of Computer Science. Thanks are also due to tmy officemates, particularly Dr Frank Zeyda and Dr Chris Marriott for their guidance and knowledge of Safety-Critical Java and formal methods. My thanks also go to Dr Tom Gibson-Robinson of the University of Oxford, UK, for giving up three days to help me wrestle my CSP_M model into a form that FDR3 liked.

My eternal thanks are due to the friends who have, possibly without knowing it, kept me buoyant during the last four years of work. James Stovold is owed my thanks for many things including his help with the York Doctoral Symposium 2014, inspiring the name $\mathrm{T}^{\mathrm{ight}}\mathrm{R}^{\mathrm{ope}}$, some light proof-reading, and some heavy sarcasm. Grace Wood, Siobhan Callaghan, Victoria Speers, and Jed Meers are each owed my gratitude for all of our nights out and nights in together. I'd also like to thank David Zendle for his help, friendly conversation, and for getting me into this mess in the first place.

I'd like to mention the dearly departed Deramore Arms, and its staff, for being a welcome place of relaxation during my Master's degree and most my of PhD. I'd also like to thank, Natalie Welden, Anna McLeod, and Scott Doyle for putting up with me during the latter part of my PhD. Finally, I must thank my parents, without whom I would not be here at all.

'The road goes ever on and on, down from the door where it began. . . '

— J.R.R. Tolkein

Declaration

I declare that the research described in this thesis is, except where stated, the original work of the author. The work was undertaken at the University of York during 2012—2016. This work has not previously been presented for an award at this, or any other, university. All sources are acknowledged as references. Parts of this thesis have been published in conference proceedings and journals; where items were published jointly with collaborators, the author of this thesis is responsible for the material presented here. We list these publications below.

Publications

[1] Matt Luckcuck. A Formal Model for the SCJ Level 2 Paradigm. In Aichernig and Bernhard Alessandro, editors, *Doctoral Symposium of Formal Methods 2015*, pages 45–48, 22 June 2015.

[2] Matt Luckcuck, Ana Cavalcanti, and Andy Wellings. A Formal Model of the Safety-Critical Java Level 2 Paradigm. In Erika Ábrahám and Marieke Huisman, editors, *Integrated Formal Methods*, Lecture Notes in Computer Science, pages 226–241. Springer International Publishing, 1 June 2016.

[3] Matt Luckcuck, Andy Wellings, and Ana Cavalcanti. Safety-Critical Java: Level 2 in Practice. *Concurrency and computation: Practice and Experience*, 2016.

[4] Andy Wellings, Matt Luckcuck, and Ana Cavalcanti. Safety-Critical Java Level 2: Motivations, Example Applications and Issues. In *Proceedings of the 11th International Workshop on Java Technologies for Real-time and Embedded Systems*, JTRES '13, pages 48–57, New York, NY, USA, 9 October 2013. ACM.

Chapter 1

Introduction

This chapter introduces our work and puts it into context within the literature. Section 1.1 presents the background and motivation for the work. Section 1.2 describes our objectives and includes the thesis statement. In Sect. 1.3 we summarise the contributions of our work. Finally, Sect. 1.4 describes the structure of this thesis.

1.1 Motivation

Safety-critical systems permeate everyday life; they are required to operate unfailingly and in real time. Examples include medical equipment, cars, aeroplanes, and power plants: areas in which failures can lead to ecological or financial disaster, serious injury, or death.

Of similar concern are systems such as deep-sea submersibles, automatic exploration vehicles (for example, the Mars rover), and other scientific monitoring systems. These systems are termed mission-critical and, while their failure may not risk life, it can cause large data- and financial-losses. Therefore, ensuring that both safety- and mission-critical systems operate as intended, in a safe and robust manner, is fundamentally important.

Java is not traditionally associated with safety-critical programming; its useful abstractions often reduce control over the predictability of execution. For example, Java's garbage collection automatically reclaims the memory used by unreachable objects, but this can delay program threads. General purpose languages are often restricted for safety and extended for expressiveness to make them amenable to safety-critical programming.

Java has features useful for safety-critical programming: a strong type system; object references, which are easier to use safely than pointers; a precise language definition; threading and synchronisation, which allow the expression of common real-time abstractions using first-class constructs; and exception handling. Java also has a wide user base, so the standard language is well understood. Finally, Java has a large variety of implementations and libraries, and is highly portable, which allows for a comprehensive range of target platforms.

Despite its useful features, Java is not expressive enough and needs restriction and extension to make it more suitable for safety-critical programming [88]. Java lacks explicit support for condition variables and the Java Virtual Machine (JVM) can spuriously wake suspended threads. Java does not support absolute time delays or detecting if a thread has resumed due to a time out (as opposed to it resuming due to being woken). Threads must hold a lock on an object before suspending on it, and when a thread resumes it attempts to reacquire the lock. However, no preference is given to resuming threads over threads attempting to gain a lock for the first time. Further, threads that suspend while holding multiple locks only release the lock on the object that they are suspending on. Workarounds for these problems can lead to less efficient executions, race conditions, or deadlocks. Finally, Java's support for scheduling and priorities is not comprehensive enough for real-time programming.

Java Specification Requests (JSRs) are a process for adding features to Java. JSR 1 produced the Real Time Specification for Java (RTSJ) [8], which introduces: real-time abstractions; region-based memory, to obviate garbage collection delays; and better control over memory usage and scheduling. The RTSJ has several implementations [3, 44, 45], but its programming paradigm is rather complex for safety-critical programs.

An international effort, led by The Open Group in JSR 302, produced Safety-Critical Java (SCJ) [79] to take the final step in creating a real-time Java language that is suitable for safety-critical programs. SCJ is aimed at applications that must be certified, for example, against standards like DO-178C/ED-12C[30]. To do this, SCJ builds on the RTSJ and adopts a new programming paradigm that is simpler to use for constructing safety-critical programs.

SCJ uses the real-time constructs and region-based memory, introduced in the RTSJ, avoiding the problems with garbage collection mentioned above. The memory areas are arranged hierarchically; associated safety rules prevent dangling references. Scheduling is achieved in SCJ with a fixed-priority scheduler that uses Priority Ceiling Emulation. These features require specialised virtual machines [72, 75] to run SCJ programs.

SCJ is organised into three compliance levels, which increase in complexity from Level 0 to Level 2. Each level defines a set of features, which include and expand on all the features of the level below. This aids certification by controlling the complexity of SCJ; if you are certifying a Level 1 program you need not worry about any of the features defined at Level 2.

Compliance Level 0 only allows simple single-processor programs that adopt a cyclic executive scheduling model. Level 1 introduces concurrency and less-restricted release patterns. Level 2 is the least restricted, compliance level. Level 2 programs are highly concurrent, potentially multi-processor, and allow suspension and a variety of release patterns.

SCJ has received attention from both industry and academia, but mostly aimed at Levels 0 and 1. Case studies using SCJ include: a port of the CDx benchmark to SCJ Level 0 [64]; an implementation of a cardiac pacemaker in SCJ Level 1 [74]; and a 3D printer, with control software written in SCJ Level 1 [76]. However, to our knowledge, there is only one implementation of Level 2: provided by the Icelab SDK[46]. Currently, the only examples of applications that exercise Level 2's features come from [89], which presents a simplified space craft and discusses a railway control system from [43].

The SCJ Language Specification does not cover techniques to verify SCJ programs; providing support for this has been left to the community. Available development and verification tools provide support for checking: memory safety [57, 24], memory consumption [4], execution time [53], schedulability [7], and functional correctness [53, 93]. These tools are discussed in Sect. 2.5, however they are also mostly aimed at Levels 0 and 1.

SCJ provides an interesting new programming paradigm, but support for Level 2 remains an open problem. The features of Level 2 are well defined in the language specification, but their utility is not well understood. Further, Level 2 example applications are difficult to find. Tools that support SCJ development are rarely explicitly aimed at Level 2, and there are no techniques for providing program verification for Level 2 programs.

1.2 Aim, Objectives, and Thesis Statement

The aim of this thesis is to support safety-critical programming in SCJ Level 2, by investigating the utility of the features provided by Level 2, and devising a model of SCJ Level 2's programming paradigm and programs. Our models are written in the state-rich process algebra *Circus*, which combines Z and CSP, guarded commands and refinement. Extensions to *Circus* provide features for capturing objects and time. These features make *Circus* a useful language for modelling object-oriented real-time languages, like SCJ.

Our first objective is to investigate the structure and execution of SCJ Level 2 programs and identify use cases where Level 2, as opposed to Levels 0 or 1, should be used. Ensuring safe use of Level 2 programs is difficult without an understanding of how its API should be used. However, example applications for Level 2 are sparse. We examine the features of SCJ Level 2 and provide example applications for which Level 2 is suited. This is the first such investigation of SCJ Level 2 programming practice.

The next objective of our work is to develop a *Circus* model of the SCJ Level 2 paradigm, as described in the SCJ language specification. This has been achieved for SCJ Level 1 [93] but not for Level 2. Our model captures the generic behaviour of SCJ Level 2, allowing it to be reused in the model of any Level 2 program. We do not just add Level 2 features to the Level 1 model; we also extend the coverage of Level 1 features.

We model the generic paradigm behaviour of SCJ separately from the program-specific behaviour of particular applications. By focussing on SCJ Level 2, we provide impetus for the development of more tools and techniques that are designed to make the use of Level 2 safer. In modelling SCJ Level 2, we also illustrate the utility of *Circus*.

Our model of the SCJ Level 2 paradigm abstracts away from scheduling and resources. This means that the model does not capture the global multi-processor support, scheduling, or region-based memory management present in Level 2. Section 2.5 discusses techniques for the verification of various program safety criteria, including schedulability and memory safety, and how suitable these techniques are for SCJ programs. In Sect. 6.3 we discuss the potential for extending our model to cover these features.

Our third objective is to develop a translation strategy to capture the program-specific behaviour of a Level 2 program and generate a *Circus* model representing its behaviour. The combination of this model with the generic model of the API provides a full specification of the program. Capturing the program-specific behaviour separately from the API reduces the complexity of both the translation strategy and application models.

Our final objective is to mechanise the translation strategy and build an automatic translation tool to produce *Circus* models of SCJ Level 2 programs. These artefacts improve the utility of our modelling approach and increase our confidence in its validity.

With the stated aim and these objectives, our thesis statement is:

The paradigm embedded in SCJ Level 2 provides features unique (within SCJ) to Level 2, which support useful applications that Levels 0 and 1 do not. Further, the

Level 2 paradigm can be formally modelled using a language that captures state and behaviour, in order to show that neither the SCJ paradigm nor a given Level 2 program introduce undesirable program states such as deadlock or divergence.

1.3 Contributions

The work we present in this thesis provides three contributions:

1. An examination of the utility of the features of SCJ Level 2,
2. A formal model of the SCJ Level 2 API, and
3. A strategy to translate SCJ Level 2 programs into our model.

Our first contribution is the examination of the utility of the features provided by SCJ Level 2. The SCJ language specification describes in detail what constitutes a Level 2 program, but not why Level 2 should be used. We provide the first examination of the features, some of the first public example applications for which Level 2's features are suited, and propose improvements to Level 2.

Our second contribution is a formal model of the API of SCJ Level 2. We model the programming paradigm described in SCJ's (natural-language) specification. This is the first formalisation of Level 2, though a model exists for Level 1. This model is generic and reusable for models of any Level 2 program.

Our final contribution is a translation strategy that captures the application-specific behaviour of SCJ Level 2 programs and constructs a model for them. This strategy gives a behavioural semantics to SCJ Level 2 in denotational style.

Our semantics combines the model of the API and the output of the translation strategy to form a model of the whole program. This complete *Circus* model can be used to support top-down and bottom-up verification of programs. In a top-down approach, our model is a target for a correct-by-construction technique for building Level 2 programs. In a bottom-up approach, our model can be used as a verification tool.

1.4 Thesis Structure

The outline of the rest of this thesis is as follows. Chapter 2 describes the details of SCJ and presents the example applications we use in later chapters. It places SCJ in context alongside other languages used for safety-critical systems, identifying their strengths and weaknesses as indicated by several programming language standards and guidelines. It introduces *Circus* and discusses similar notations. Finally, it places this work in the wider context of verifying SCJ programs by describing the tools available for SCJ program verification.

Chapter 3 examines the utility of the features found in SCJ Level 2 and presents some example applications for which Level 2 is particularly suited. This is an essential step in understanding the sorts of programs for which Level 2 is likely to be used.

Chapter 4 presents our approach to modelling the SCJ Level 2 API and Level 2 programs. Further, we describe how our model captures the more complex situations of inheritance and polymorphism, and synchronisation and suspension.

Chapter 5 describes in detail how to construct models of Level 2 programs that are compatible with our model of the API. This translation completes our approach to providing a bottom-up technique to verify Level 2 programs, and supports the validity of our models. We present the formalisation of part of our translation strategy in Z and describe a prototype tool for automatic translation of Level 2 programs.

Finally, Chap. 6 provides a summary of the thesis contributions. It also discusses the utility and validity of our modelling approach and translation strategy. Finally, it discusses future work that builds on our approach.

Chapter 2

SCJ, *Circus*, and Verification

This chapter introduces Safety-Critical Java (SCJ), the features that make it a useful safety-critical language, and its applicable verification techniques. Section 2.1 describes SCJ's features in detail. In Sect. 2.3 we discuss the utility of SCJ as a safety-critical language. Section 2.4 introduces *Circus* and compares it to similar notations. In Sect 2.5 we describe the current verification techniques that are applicable to SCJ programs, including a technique based on *Circus*. Finally, Sect. 2.6 summarises the main points raised in the chapter and their impact on our work.

2.1 Safety-Critical Java

In this section we describe SCJ in detail. Section 2.1.1 provides a full overview of SCJ's features. We discuss the structure of SCJ programs, memory management, and compliance levels. However, as mentioned in Sect. 1.2, the model that we present in Chap. 4 abstracts away from scheduling and resources. So, Sect. 2.1.1 describes features of SCJ Level 2 that are not covered by our model. Finally, Sects. 2.1.3 and 2.1.2 provide example applications written in SCJ, which we reuse in later chapters.

2.1.1 SCJ Overview

SCJ is a version of Java that adopts a new programming paradigm to support the development of software that must be certified. It uses region-based memory management to avoid garbage collection delays and improve control of memory usage; rules prevent dangling references. SCJ also provides common real-time abstractions to support tasks. These features mean that SCJ programs require specialised virtual machines [72, 75].

SCJ borrows its real-time abstractions and region-based memory management from the Real-Time Specification for Java (RTSJ), which is a version of Java designed for real-time systems. However, its programming paradigm is rather complex, which makes it less suitable for constructing safety-critical programs. SCJ takes the real-time elements of the RTSJ and provides its own, restricted, programming paradigm to make it easier to construct safe and certifiable programs.

The SCJ API relieves the programmer of most of the burden of adhering to SCJ's programming paradigm by providing its generic behaviour and structure. This collection of classes and interfaces must be overridden or implemented to construct a program.

Name	Description
`Safelet`	Controls the whole infrastructure and starts the `MissionSequencer`
`MissionSequencer`	Instantiates and starts a sequence of `Missions`
`Mission`	Controls a set of tasks, represented by subclasses of `ManagedSchedulable`
`ManagedSchedulable`	The super-interface of all schedulable objects
`PeriodicEventHandler`	A schedulable that executes once every period
`OneShotEventHandler`	A schedulable that executes once after a time offset
`AperiodicEventHandler`	A schedulable that executes when triggered by a method call
`ManagedThread`	A schedulable that executes immediately, when the mission starts

Table 2.1: SCJ Paradigm Components

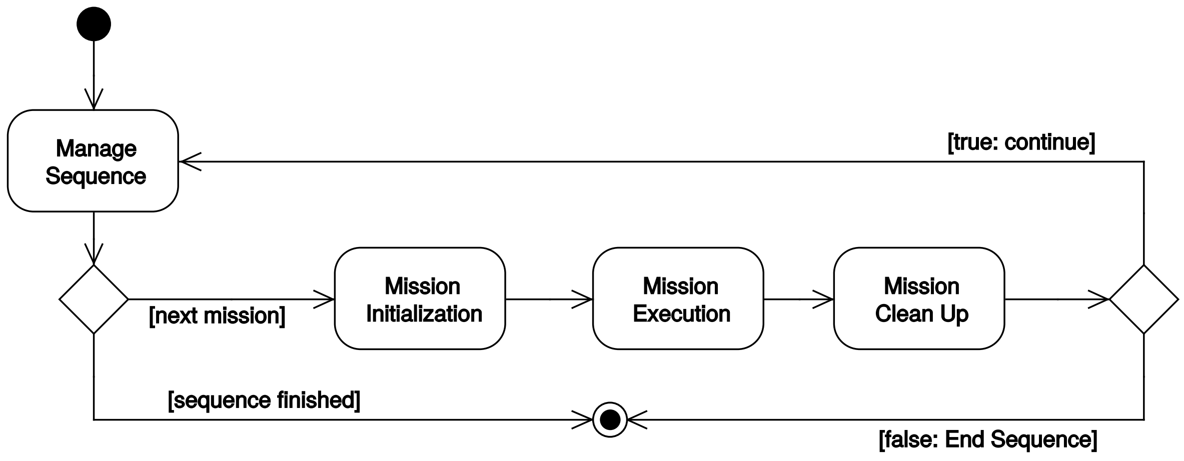


Figure 2.1: SCJ Mission Phases, from [79]

At the top of every program hierarchy is a *safelet*, which controls the entire program. The safelet chooses and starts a *mission sequencer*, which defines a sequence of *mission*s to be executed. The mission is the key component in structuring SCJ programs; each mission encapsulates a function of the system and controls a set of tasks to achieve the required behaviour. In SCJ, tasks are called *schedulable objects* and can take one of four release patterns that we describe later. These paradigm components are summarised in Table 2.1.

Each mission progress through three phases, as shown in Fig. 2.1. First, in the *initialisation* phase, it registers its schedulable objects and initialises any data structures it requires. Next, in the *execution* phase, each of the registered schedulables begins executing. Termination of a mission is triggered by all of its registered schedulables finishing or at the request of one of its schedulables. During termination, the mission terminates each of its schedulables, then the mission enters the *clean up* phase, where it can reset any changes it made to shared resources. At the end of the clean up phase, the mission sends a signal to indicate if its controlling mission sequencer should load another mission or not.

Although real-time Java garbage collection mechanisms are available [73, 71], SCJ uses a hierarchical and region-based memory model that avoids heap use. The memory hierarchy is

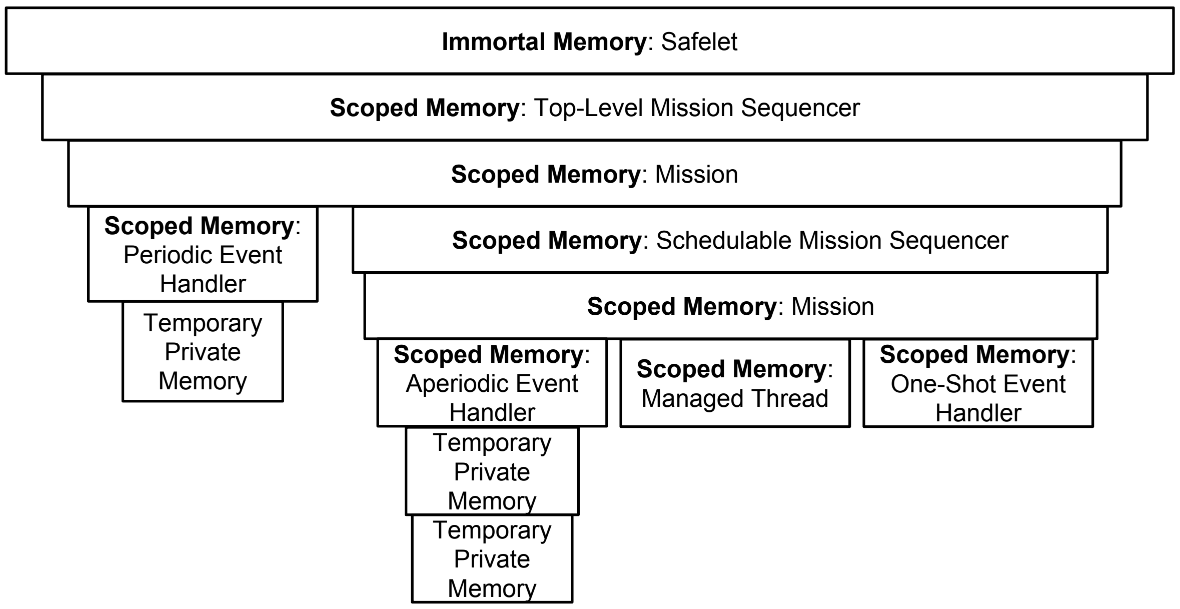

Immortal Memory: Safelet
Scoped Memory: Top-Level Mission Sequencer
Scoped Memory: Mission
Scoped Memory: Periodic Event Handler
Temporary Private Memory
Scoped Memory: Schedulable Mission Sequencer
Scoped Memory: Mission
Scoped Memory: Aperiodic Event Handler
Temporary Private Memory
Temporary Private Memory
Scoped Memory: Managed Thread
Scoped Memory: One-Shot Event Handler

Figure 2.2: SCJ Memory Hierarchy

illustrated in Fig. 2.2, where each box is a memory area and the bold text shows the type of memory area: immortal memory or scoped memory. Each SCJ API class has a memory area in which its allocations are made by default, the default allocation context, which is shown in Fig. 2.2 after the bold text. Other memory areas can be entered during execution. The immortal memory area is the top-level memory area, which persists for the lifetime of the application. It is the default allocation context for the safelet.

Each scoped memory area persists for the lifetime of the component with which it is associated. Each mission has its own scoped memory area, called mission memory, which is cleared at the end of its clean up phase. Each schedulable has a scoped memory area that is cleared after its release; this includes the mission sequencers, which are a subclass of `ManagedEventHandler`. Data that is shared between schedulables must be stored in a mission memory area or in immortal memory.

Temporary private memory areas are a scoped memory area that may be entered during a mission's initialisation or by schedulable objects during their release. This changes the default allocation context, to the temporary private memory area, until the execution running inside it terminates, which triggers the clearing of the memory area.

In an SCJ program, these memory areas form a hierarchy (shown in Fig. 2.2) where each memory area uses some of the space reserved for its parent. At the top of this hierarchy is immortal memory, followed by the private memory area of the mission sequencer loaded by the safelet: the top-level mission sequencer. Next is the mission memory area used, in turn, by each mission loaded by the top-level mission sequencer. As new components become active, their associated memory areas are added to the hierarchy; when a component terminate, its memory area is removed.

Figure 2.2 shows the memory hierarchy of a program that has a mission that is running a periodic event handler and a schedulable mission sequencer. The periodic event handler is active and has entered a temporary private memory area. The schedulable mission sequencer has loaded a mission, which is running three schedulable objects. One of these schedulables,

Level	Unique Schedulables	Nested Missions	Suspension	Processor
Level 0	Periodic	No	No	Single
Level 1	Aperiodic, One-Shot	No	No	Multi
Level 2	Managed Thread, Mission Sequencer	Yes	Yes	Global Multi

Table 2.2: SCJ Features by Compliance Level

the aperiodic event handler, has (sequentially) entered two temporary private memory areas. These memory areas are nested, indicating that the space for the second temporary private memory area is taken from the space reserved for the first.

Memory areas further down the hierarchy have a shorter lifetime. To avoid dangling references, a reference may only point to an object stored in the same memory area or in a memory area that is further up the hierarchy. That is, references can only point to objects in memory areas that are cleared after, or at the same time as, the memory area that contains the reference. We note that our model in Chap. 4 abstracts away from SCJ memory management. Techniques for checking adherence to the SCJ memory safety rules, to show that a program is memory safe, are discussed in Sect. 2.5.

SCJ is organised into three compliance levels, which ascend in complexity from Level 0 to Level 2. Our work is aimed at Level 2, which is the most complex, or least restricted, compliance level. Each compliance level provides some unique features, while maintaining the features of the level(s) below it. The features available at each compliance level are summarised in Table 2.2 and described in detail below.

Level 0 is for sequential programs that adopt a cyclic executive, where a set of computations are executed periodically. Level 0 programs only contain a single mission sequencer, but may use multiple (sequential) missions. A Level 0 program's schedulable objects may only consist of *periodic event handler*s, which execute their behaviour periodically after a time offset. At Level 0, the scheduling of programs is restricted to one processor only. An example of a Level 0 program is a simple aircraft collision detection application [92].

Level 1 programs contain only one mission sequencer and may use multiple sequential missions, as at Level 0. However, a Level 1 program's schedulable objects my include periodic event handlers, *one-shot event handler*s, and *aperiodic event handler*s. A one-shot event handler executes its behaviour once, after a time offset. An aperiodic event handler has no set release time; it executes its behaviour when requested by a method call. Further, a specialised type of aperiodic event handler, called an aperiodic long event handler, takes a `long` parameter to allow information to be passed to the event handler during execution.

At Level 1, schedulables are concurrent. Active schedulables may preempt each other based on their respective priorities. Access to shared data can be controlled with synchronised methods, to avoid race conditions; synchronised blocks are not allowed in SCJ. At Level 1, multiple processors may be used, but each schedulable may only be assigned to one processor. An example of a Level 1 program is a cardiac pacemaker [74].

Level 2 is the most complex compliance level, suitable for highly concurrent programs. Level 2 programs may use the three event handlers available at Level 1, and *managed thread*s.

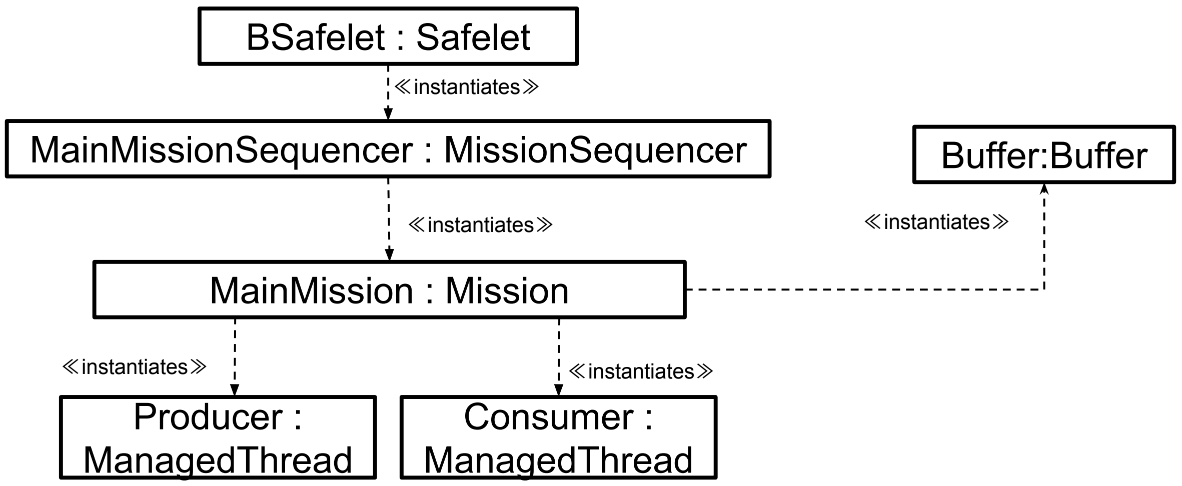


Figure 2.3: Object Diagram of the Buffer Example Application

A managed thread is a real-time thread that is released immediately, executes it behaviour, and then terminates. Level 2 programs may contain multiple mission sequencers, each registered to a mission. This is possible because a mission sequencer is a schedulable object. This feature allows a Level 2 program to have multiple active missions. However, there may only be one active mission per mission sequencer; the maximum number of active missions is equal to the number of active mission sequencers.

Level 2 programs are able to use a restricted version of Java's suspension model. The `Object.wait()` and `Object.notify()` family of methods may be used, but they may only be invoked on `this`. That is, the target of the call can only be the current object – the object containing the code currently being executed. Further, all queuing threads wait in eligibility order. The most eligible waiting thread has the highest priority and has been waiting the longest. Finally, Level 2 programs may be scheduled globally over multiple processors. That is, the scheduler may select from several processors when executing a schedulable object. Our work is concerned with the SCJ paradigm; since scheduling is handled by the SCJ virtual machine, we do not discuss this feature further.

Below, we present two example applications to show the utility and complexity of Level 2. The first is a simple producer-consumer program, the second is a simplified aircraft control system. Further examples of applications that are supported by Level 2's features are discussed in Chap. 3.

2.1.2 Buffer: A Producer-Consumer Application

In this section we present a buffer application, which is a simple solution to the Producer-Consumer problem. This basic example illustrates how to compose an SCJ program, and shows the use of managed threads and suspension. The full program listing can be found in Appendix A.

An object diagram of the buffer application is shown in Fig. 2.3, where the arrows represent instantiation of the target object. The application contains a buffer object, which is instantiated by the mission during its initialisation phase. Two managed threads, one producer and one consumer, share access to this buffer. Access to the buffer is controlled using suspension. The producer suspends if the buffer is full; if not, then it writes and notifies the

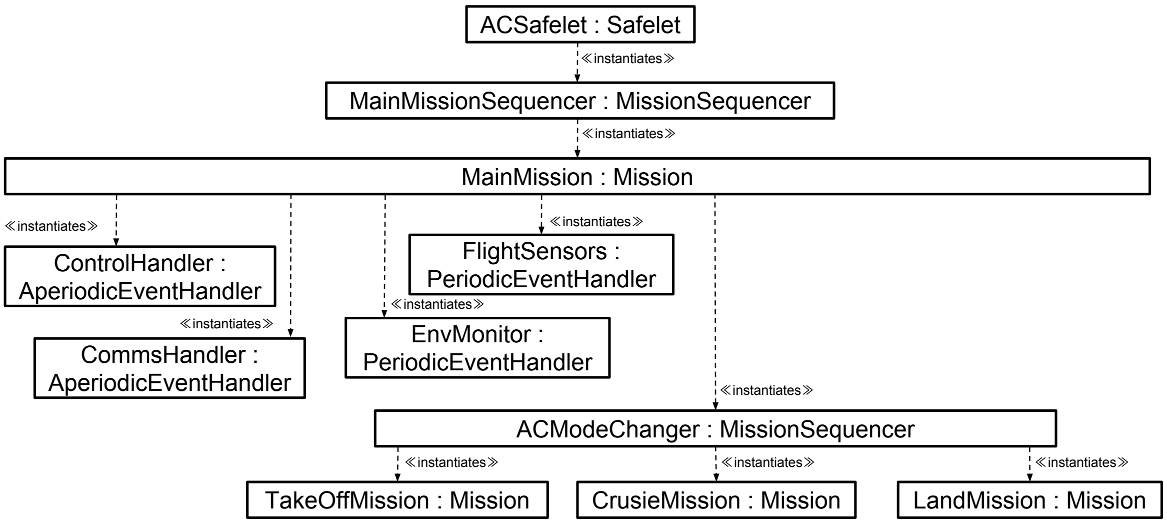


Figure 2.4: Simplified Object Diagram of Aircraft Example Application

consumer. The consumer suspends if the buffer is empty; if not, then it removes a value from the buffer and notifies the producer. Suspension is achieved with a call to `Buffer.wait()`. Notification is achieved by a call to `Buffer.notify()`. After reading from the buffer 5 times, the consumer requests that the mission terminates. When both the managed threads have terminated, the program terminates.

2.1.3 Aircraft: a Multi-Mode Application

To show the complexity of Level 2's features, we present a simplified aircraft control program as an example application. This example is adapted from one in [89] that shows the complexity of concurrent missions in Level 2. The full program listing can be found in Appendix B.

Our simplified aircraft has three modes of operations, which correspond to the phases of flight: Take Off, Cruising, and Landing. Each of these has mode-specific behaviours that are only pertinent during that phase of flight. The aircraft also has persistent behaviours that are pertinent during all modes. A simplified object diagram of the aircraft control application is shown in Fig. 2.4, where the arrows represent instantiation of the target object.

The program is controlled by the `ACSafelet`, which starts the `MainMissionSequencer`. The `MainMission` controls the persistent schedulables, including the mission sequencer that is used to change between modes. The missions representing the modes (`TakeOffMission`, `CruiseMission`, `LandMission`) are controlled by the `ACModeChanger`. Each of these missions controls its mode-specific schedulables, which are omitted from Fig. 2.4 for brevity.

SCJ Level 2 is particularly suited to capturing systems like the aircraft application, which have multiple modes with their own schedulables *and* schedulables that run during all modes. As discussed in Sect. 3.2.1, Level 1 can be used to program applications with multiple modes, but persistent schedulables must be duplicated because Level 1 programs cannot have concurrent missions. The duplication of the persistent schedulable disrupts their operation during mission changes, therefore Level 2 provides better control over programs with multiple modes.

2.2 Safety-Critical Standards and Language Assessment

Because safety-critical systems are often complex and their correct functioning is of such paramount importance, standards and guidelines are used to ensure that the software functions correctly. The SCJ Language Specification [79] is specifically aimed at software systems that require certification, so the understanding of what is prescribed in such standards is important. Section 2.2.1 discusses standards for safety-critical programs, and Sect. 2.2.2 assesses Ada, C and C++, and Java-based languages against these standards.

2.2.1 Standards and Guidelines

Safety-critical programming standards typically restrict the behaviour of programs, prescribe or proscribe certain language features, or specify the processes that must be followed during program development. More complicated standards combine some or all of these approaches. Some standards are specific to certain languages – Ada or C, for example – or specific sectors of industry – such as avionics or nuclear reactors – whereas others provide more general guidelines. There are standards that were designed for a specific industry but are now more widely used. For example, MISRA C [60], which was designed for the automotive industry but has gained wider use within the safety-critical community.

During the mid 1970's the USA Department of Defense (US DoD) produced the 'Steelman' requirements [84], which contains features that were used to create Ada, in 1983. The requirements are grouped as either application, environment, or commonality features. The application requirements are the ability to specify user interfaces, exception handling features, real-time control, and the ability to perform parallel processing. The environmental requirements are that programs have to be reliable, modifiable, and efficient. Lastly, the commonality requirements are that the language should be machine independent, be easy and inexpensive to implement, be completely defined, and have easily accessible support software.

The Steelman requirements have been an influential benchmark. For example, they have been used to compare Ada95, Java, C, and C++ [90]. This is useful, not only to compare versions of the Ada language, but to compare other languages used in the area of safety-critical programming based on the requirements for the creation of Ada83, which is one of the most popular languages in the domain.

The book 'Safer C', aimed at making the use of C safer in safety-critical programming, contends that "it is not how safe a language is, but how safe the use of a language can be made that matters" [37]. This book compares C to other contemporary languages, including Ada83. These comparisons include non-functional features of language use, such as its user base and the availability of appropriate tool support.

Given this holistic approach to language safety, it is no surprise that Safer C accepts the flaws in C and strives to identify the potentially dangerous language features and restrict their use. The book identifies the dangerous language features: dynamic objects, because of their potential for causing memory leaks if not properly deallocated; recursion, because of its potential to exhaust the available memory; overloading operators, because of the potential confusion this brings; and inheritance, specifically multiple inheritance because it can lead to highly complex architectures. Safer C also mentions the importance of the language's grammar, in terms of reading and writing a complier for the language. Some of these features are simply prohibited, such as recursion, and others are only cautionary points, such as warning of the problems with multiple inheritance.

The comparisons that Safer C makes between C and other languages are very detailed and use varying sources. For example, language definitions, experiments, and interviews with developers are all used. However, each comparison is only made pair-wise with C. This is because the work is focussed around making C safer for use in safety-critical software. It is no surprise to find, therefore, that the book concludes that C is a better option for reasons such as its large user base (and so wider user testing), extensive libraries, and tool support.

The MISRA C standard [60] defines a subset of C for use in embedded automotive systems. It also promotes the safe use of C and raises awareness of language-choice issues generally. A further goal was to encourage the developers of commercial-off-the-shelf programming tools to ensure their tools are suitable for the automotive industry. MISRA C contains 93 mandatory and 34 advisory rules. The rules themselves are grouped into 17 different categories, which include sections on basic language features (like types, identifiers, and constants) and more complicated constructs (such as pointers and arrays). It recognises that vulnerabilities exist in C and provides a series of rules concerning the use of the language. Vulnerabilities it notes for C include: programming mistakes, like typing '==' instead of '='; programmer misunderstanding, for example, of the operator precedence in the language; compiler errors; and run-time errors. MISRA C also recognises that language vulnerabilities are only a small part of program safety, and so it gives guidance for 'best-practice' development processes.

The DO-178/ED-12, "Software Considerations in Airborne Systems and Equipment Certification", series of civil avionics standards have been very influential in safety-critical program certification. The DO-178B/ED-12B [29] version of the standard was published in 1992 and adopted in Europe and North America; it is concerned with the safety of software-controlled systems in the air. The DO-178/ED-12 is specifically mentioned in the SCJ Language Specification as an example of the level of certification at which SCJ is aimed. It presents 66 guidelines for programming software safely and specifies a series of outputs from the development process, which provide enough evidence that the software has been built correctly. It intentionally does not emphasise any particular development method, so that certification can be applied to a wide range of development processes.

The DO-178B/ED-12B has been adopted in areas other than civil avionics. The NASA Software Safety Guidebook uses the definition of certification from the DO-178B/ED-12B and a NASA report on certification for safety- and mission-critical software essentially adopts the standard, adding extra details. The standard is also referenced in two standards adopted by the US military. The MIL-HDBK-516B uses DO-178B/ED-12B as a reference in its section on software safety, along with two other documents, and the Software System Safety Handbook makes reference to the DO-178B/ED-12B amongst others [52].

In 2012 the DO-178B/ED-12B was updated, the new DO-178C/ED-12C [30] contains updates in vocabulary and concepts, to bring them into line with current usage and the state of the art. For example, the phrase 'target computer' was felt to be outdated and was changed to 'execution platform' [49]. This accommodates the use of virtualisation in which the phrase 'target computer' is ambiguous. The updated standard gains four technology supplements, including one for Object Oriented Programming [69] and one for Formal Methods [68] to provide guidance for prevalent technologies in the safety-critical programming domain.

The DO-178C/ED-12C Object-Oriented Technology Supplement [69] presents the vulnerabilities of using OO programming languages and describes how they may be dealt with. The supplement aims to resolve the problems with the features core to most OO languages. There is also a section discussing heap memory management in which real-time garbage collection is allowed for the first time. The Formal Methods Supplement [68] provides guidance for their use in the development of the software to be verified by the standard. This does not mean that testing can be eliminated, but the standard does mention that appropriate use of formal methods can reduce the burden on program testing.

The inclusion in the DO-178/ED-12 family of standards of the topics of virtualisation and garbage collection is particularly useful for Java-based languages in avionics, and in safety-critical software generally, given that they generally run on a virtual machine and use automatic garbage collection [42]. The guidance provided in the Formal Methods Supplement also adds credence to the proposed development of formal methods for SCJ Level 2.

2.2.2 Safety-Critical Language Assessment

There have been profiles and subsets of Ada, C and C++, and Java and the RTSJ that improve on the ease of producing a safe program of the base language. Given the popularity of these languages and their subsequent adaptations there has been much discussion in the literature regarding their safety. The results of assessing these languages against various standards and their pertinent safety features are presented below.

Ada

As previously mentioned, Ada was constructed from the 'Steelman' requirements [84], which are grouped into application, environment, and commonality features. Unsurprisingly, Ada95 was found to meet 93% of the Steelman requirements [90]. However, this shows that Ada95 still satisfies most of its original requirements. But, the relevance of the Steelman requirements is questionable, as they have not been updated and so do not include features regularly found in modern programming languages – such as object orientation.

When comparing the suitability of Ada83 and Ada95 for use in safety-critical systems, Ada95 was found to be an improvement on the safety of Ada83 [22], which again is no surprise. The analysis used a framework of 10 categories of features, organised into four sections: predictability, analysability, traceability, and engineering concerns. Each feature in the Ada Reference Manual for each version of the language was assessed against this framework. However, this does not represent a contemporary view of Ada.

Ada83 has also been compared to assembly languages, C, and three other languages, along with their subsets and possible subsets [23]. Each language was assessed against a set of questions designed to elicit the insecurities of a programming language when it is used in safety-critical systems. This list includes questions regarding: wild jumps, memory overwrites, well defined semantics, strong data typing, exception handling, and the language being well understood. This analysis showed that, at that time, a subset of ISO Pascal was the safest choice out the languages assessed. But a hypothetical Ada subset – based on work like the foundations of SPARK Ada [14] – was the next safest choice, though this is based purely on theoretical considerations.

A comparison between Ada95 and both standard Java and the RTSJ [70] found that Ada95 provides much better concurrency control; a range of pitfalls in the Java concurrency model can be solved or prevented with the use of Ada95. However, Java was not designed for hard real-time programming and has had to rely on the RTSJ to remedy problems in the Java concurrency model, for example the delays caused by garbage collection. Despite this, the study concludes that Java is 'adequate' for safe real-time programming, but that Ada95 provides better confidence in the safety of the real-time systems it can construct. This is because Ada provides particular features designed for real-time safety such as: protected objects, which have an associated monitor and locks on all their operations; the provision of condition synchronisation; the resumption of the most eligible waiting thread; and the prevention of an operation blocking while holding a mutually exclusive lock.

Ada is unique in having a strong type system, a wide range of static types, a consistent semantics defined in an international standard, support for abstraction, and validated compilers. This makes makes it well suited for safety-critical applications programming. However, to achieve full predictability, static analysis, and testing of programs written in Ada, it may still be necessary to restrict or control certain features within the language [47]. Due to its popularity and safety features, Ada subsets have been developed. These subsets, SPARK [15] and Ravenscar [28], for example, aim to improve on the already stable base provided by standard Ada.

The SPARK Ada subset [15] provides a set of annotations to add extra semantic information to a program. Not only are these useful when it comes to the human-readability of program source, they also present hooks for static analysis. Tools – such as the SPARK Examiner – can use this embedded information to check that the program performs as the annotations dictate, using the annotations as fragments of specification embedded in the program itself. However, using annotations like this adds an overheard to the development process. SPARK Ada also provides a much simpler programming model, when compared to Ada, while still maintaining the expressiveness needed for real-time programs. This has the dual-purpose of improving a program's amenability to verification tools and improving the likelihood that the programmers intentions will be encoded into the program correctly.

C and C++

C is still commonly used in safety-critical systems; there are reports of C being used even in situations where Ada has been mandated or is required [37]. When assessed against the Steelman requirements, C was 53% and C++ was 68% compliant, neither coming close to the 93% that Ada95 scored on the same assessment [90]. C failed an assessment against a set of criteria for languages in safety-critical systems and so its use in such systems is not advised [23]. However, Safer C [37] contends that this study was done before the C standard was finalised and because it did not include tool support, its assessment of C was too heavy-handed. Summarising both [37] and [55], [36] lists some of the difficulties with using C for safety-critical programs, for example: weak typing, dynamic memory allocation, pointer arithmetic, undefined data types, potential ambiguities in `if` conditions, and increment or decrement operators.

Safer C [37] takes the opinion that the potential for making a language safe to use is more important than its intrinsic safety. Safer C [37] considers the wide user base of C in making an assessment of how safe it is, realising that this needfully brings a wide understanding of the language. Safer C also takes an holistic view of the software being developed, in that tool support, language profiles or subsets, and the wide user base, are all factors affecting the development process and therefore all contribute the safety of the final program.

C was compared pair-wise with Ada83, C++, FORTRAN, and Modula-2, which concluded that, while areas of Ada83 are fundamentally safer, the tool support for C is so good that in places it can match or exceed this safety [37]. This is a useful point: assessing the intrinsic safety of a language does not provide a full view of the safety of its programs, assessing a language coupled with its prevalent tool support provides a fuller picture of language suitability for safety-critical programming. If commonly used tools or a particular language subset provide a safety feature, it is likely that this will have a positive effect on the safety of the software produced.

Java and the RTSJ

Java has long been dismissed for use in safety-critical programs due to its root as a general-purpose programming language. Though Java's automatic garbage collection prevents accidental or forgotten deallocation, it also reduces the predictability of a running program because program threads may be delayed by the garbage collector. Java has some redeeming features, for example its strong typing, precise definition, and the inclusion of language features like exceptions [40]. When assessed against the Steelman requirements, Java 1.0 met 72% of them [90]. Java 1.1 was assessed [41] in terms of defects found in the language, using evidence from experiments, the Java language specification, web sources, and published research. Java was found to have fewer defects, and therefore be far safer, than both C or C++. However, this study also assessed Ada95, which had the fewest defects.

Java 1.5 was assessed against a conglomerate framework of several safety-critical standards [48]; this assessment concluded that the language has some weaknesses and, while standard Java is very useful for general purpose programming and contains several useful features, it is still not appropriate for high-integrity systems. This assessment also concludes that safety-critical subsets that use Java as their base could be developed. Coupled with the development of formal analysis techniques – for example, model checking – such a subset could make Java a viable option. This work resulted in a subset of the RTSJ, which uses Ravenscar Ada as its template and focusses on reliability, called Ravenscar-Java.

The RTSJ was aimed at addressing some of the problems of standard Java and making it amenable for use in real-time software. It introduces memory scopes, which are not subject to automatic garbage collection, and improves support for handling interrupts [70]; it also increases the control programmers have on the scheduling of concurrent activities [88]. RTSJ also succeeds in preventing dangling references though its memory allocation rules and provides explicit support for paradigms like asynchronous or periodic events [9].

The features introduced in the RTSJ add expressiveness to the language, but certain pitfalls are retained from standard Java. These include problems like the ability to program `synchronized` blocks of code within methods and the lack of explicit support for condition variables. While the scoped memory system of the RTSJ does improve the predictability of programs, because all scopes other than the heap are not subject to garbage collection, it is complicated and requires common programming paradigms to be rethought [9].

However as these problems are well known they can be overcome; for example, development tools that are aware of the problems can help a programmer identify when they might occur. Real-time Java profiles remain an attractive prospect for a safety-critical system where Java is the chosen technology [9]. With specific regard to problems encountered due to the complexity of the RTSJ scoped memory model, programming patterns have already been developed that either adapt current programming patterns to be scope-aware – like the adapted Singleton Pattern or Memory-Aware Factory Pattern [21] – or use memory scopes more effectively – for example the Handoff Pattern [63].

The next section provides a detailed discussion of the relationship between the features of SCJ and safety-critical standards. It provides an analysis of how easily certifiable SCJ programs are, under the categories of predictability and reliability, analysability, and pragmatic design.

2.3 SCJ and Safety-Critical Standards

In this section we discuss how the features of SCJ improve the certifiability of its programs. We combine the categories given in several safety-critical programming standards and guidelines to frame our discussion. We address how SCJ improves the predictability and reliability (Sect. 2.3.1), analysability (Sect. 2.3.2), and pragmatic design (Sect. 2.3.3) of its programs.

Safety-critical programming languages should focus on providing features that allow program verification because it is the main prerequisite for the certification of software-systems [36]. Certain language features can frustrate program verification, and are often prohibited or controlled by safety-critical standards.

Given that Ada was created with safety-critical systems in mind, we consider the ISO/IEC technical report 'Guide for the use of the Ada programming language in high integrity systems' [47], which focusses on the control or removal of language features in Ada that prevent program verification. It considers four motivations for rules that require or reject a particular language feature: predictability, testing, modelling, and pragmatism.

- Rules to achieve program predictability strive to ensure that program code is unambiguous. These rules, which are separate from the analysis methods, include side-effects in functions and evaluation-order effects.

- Rules to facilitate testing aim to remove any features that might prevent the verification of dynamic program behaviour. These features include constructs that disrupt the program flow or complicate the view of the system state.

- Rules to aid modelling are concerned with control of language features that are difficult to model or produce intractable models. These features include aliasing of objects, parameters, and other identifiers; and features causing complicated execution – like recursion and concurrency. The report considers both formal and informal modelling.

- Pragmatic considerations deal with features that promote good program design. These features are important because the architecture of the application, variable scope, visibility, and so on, affect how easy it is to relate the program to its specification.

The report also considers additions to the language that can aid verification. For example, loop invariants and hidden state, which are understood by the programmer, but are not expressed in the program. This additional information can particularly aid formal verification, and often takes the form of program annotations, like those present in Spark Ada or SCJ, to embed the information within the program.

The DO-178C/ED-12C ('Software Considerations in Airborne Systems and Equipment Certification') civil avionics standard [30], and its previous versions, have been very influential in safety-critical program certification. It is specifically mentioned as a standard at which SCJ programs are aimed. Its predecessor (DO-178B/ED-12B [29]) has been adopted in several areas other than civil aviation [52]. The DO-178/ED-12 standards contain four themes: reliability, predictability, analysability, and expressiveness.

DO-178C/ED-12C has four technology supplements, adding guidance for particular software development technologies. Crucially for our work, these include a supplement for Object-Oriented Programming [69] and for Formal Methods [68]. This aids the certification of SCJ programs and formal methods for safety-critical development.

Craigen et al [22] construct a framework for comparing the suitability of different versions of Ada for use in safety-critical systems. It draws on the DO-187B/ED-12B, British Ministry of Defence MD00-55 and MD00-56 standards, and the Canadian Trusted Computer Product Evaluation Criteria. Its features are organised into four themes:

- predictability, so that the system behaves unambiguously;
- analysability, so that language features allow tractable analysis;
- traceability, so that requirements can be tracked through the development process; and,
- engineering, features that aid the flexibility of design choices.

During the construction of an earlier Java profile for safety-critical programming, Ravenscar Java [48], the features of Java and the RTSJ were assessed against a framework combining several standards and guidelines, including DO-178B/ED-12B. The framework is organised into three broad categories, which each have two levels: mandatory and desirable.

The categories and mandatory features are:

- Syntax and Semantics, which is subdivided into
  - Type Safety and Strong Type Rules,
  - Clear Description of Side Effects and Operator Precedence,
  - Modularity and Structures,
  - Formal Semantics and International Standards,

    - Well Understood Semantics,
    - Embedded Systems Support, and
    - Concurrency;

- Predictability and Verification, which is subdivided into

    - Functional Predictability,
    - Temporal Predictability and Timing Analysis, and
    - Resource Usage Analysis; and

- Language processors, Run-Time Environment, and Tools, which is subdivided into

    - Certified Translators, and
    - Run-Time Support and Environment Issues.

The features in the framework are rather broad, but they are useful to structure the discussion that follows. Below, we discuss how the features of Java fit into the categories of predictability and reliability, analysability, and pragmatic design. Predictability and reliability are key features that are present in all of the standards discussed above. Analysability covers features that aid the testing, modelling, and verification of a program. The pragmatic design category covers features that improve the traceability, expressiveness, or engineering capabilities of the language, which includes language translators and tools.

2.3.1 Predictability and Reliability

SCJ improves the predictability and reliability of it programs over Java. It uses the region-based memory model introduced in the RTSJ, in which memory areas are deallocated by the infrastructure when there are no threads active inside the memory area. This avoids garbage collection reclaiming unused memory at an arbitrary point in time. Though there are predictable garbage collection techniques [73, 71], SCJ chooses memory areas to simplify its infrastructure. Further, the size of each memory area in an SCJ program can be defined and fixed at compile-time. The memory usage and safety are analysable, as we discuss in Sects. 2.5.2 and 2.5.3.

SCJ restricts the permissible structure of its programs, which makes their execution more predictable. As discussed in Sect. 2.1, SCJ programs are hierarchical. This structure adds a predictable instantiation order during the set up of the program. The predictable use of memory and tasks execution times during execution are discussed below.

The time predictability of SCJ tasks is achieved using the real-time constructs introduced in the RTSJ. Each of the three handler classes have release times and may have deadlines, which can be defined and fixed at compile-time. Techniques are available for Worst Case Execution Time analysis, which we discuss in Sect. 2.5.1.

The SCJ suspension behaviour is also more predictable than Java's. Threads queue for locks in order of eligibility. This provides a more predictable order for thread resumption.

2.3.2 Analysability

SCJ is designed to produce analysable programs, ensuring that they can be tested and modelled, because this aids the development of safe programs. This occurs in several areas of the language. The restrictions to SCJ's memory model allow static memory usage analysis and its simplified concurrency model aids schedulability analysis [40].

The novel SCJ paradigm is implemented on top of standard Java, which is widely understood and has a publicly available language specification. The popularity of standard Java means that there are many testing tools that can be applied to the core parts of SCJ programs. JUnit, for example, can be used to perform unit testing on the methods that implement the behaviour of the schedulable objects in a program. The `assert` statement, which is built into Java, can be used to test assertions during development.

The safety-critical features of an SCJ program, however, require SCJ-specific techniques. There are techniques available for: memory safety [57, 24], memory consumption [4], execution time [53], schedulability [7], and functional correctness [53, 93]. These tools are discussed in Sect. 2.5, however they are mostly aimed at Levels 0 and 1.

SCJ is defined informally in its language specification. However, this is an invaluable resource for understanding the purpose and meaning of language features when modelling SCJ programs. This specification has been the main source of information used in many efforts to model and test SCJ programs, particularly as a full implementation of SCJ only emerged at the beginning of 2015 [46].

SCJ has been the focus of a series of formal models written in *Circus*, which can be related by refinement to produce concrete SCJ models from abstract specifications [19]. This technique can be used to develop programs that are correct-by-construction. The SCJ memory model [16] has been captured in *Circus*. This model does not cover the control flow of programs, as this is addressed elsewhere. It is explicitly aimed at Level 1 programs: it only uses a single mission memory area, which makes it unsuitable for Level 2.

The paradigm of SCJ Level 1 programs has been modelled in *Circus* [93], which provides the refinement strategy with a target for Level 1 programs and allows programs to be model checked. We build on this work in our model of SCJ Level 2. At a level of abstraction that is closer to the programs themselves, *SCJ-Circus* [59] provides a formal notation for capturing the components of SCJ. Its syntax and semantics are defined by mapping its constructs back to standard *Circus*.

These efforts show that SCJ is analysable. Features of the language aid program testing, and the information that programs contain can be used by analysis tools and for modelling. The techniques that are available for analysing SCJ programs, however, generally ignore Level 2, so this is an open area.

2.3.3 Pragmatic Design

SCJ is based on the firm foundation of Java, which is a popular and well understood language. This makes it a good base on which to build a new safety-critical language. With such a large pool of programmers who understand Java, the jump to understanding SCJ is relatively small, compared to the jump from Java (or other C-like languages) to a language like Ada.

Further, SCJ is strongly typed and object-oriented, which means that common programming errors are detected at compile-time.

The potential of tools to prove properties about SCJ programs is greater than that for C or C++, because its semantics are less ambiguously defined. Java features like keywords, which aid readability, and only using single inheritance help here. Further, Java applications are compiled to standardised bytecode, and tools can analyse programs at this level [40].

SCJ's basis in standard Java means that porting existing programs into SCJ is easier. Further, SCJ uses common real-time abstractions such as periodic and aperiodic tasks, and threads. This, again, eases the porting of real-time programming patterns to SCJ.

SCJ's novel memory model can seem complicated, at first. Programming patterns have been adapted, for the RTSJ, to be scope-aware – like the adapted Singleton Pattern and Memory-Aware Factory Pattern [21] – or to use memory areas more effectively – like the Handoff Pattern [63]. These patterns can be applied or adapted for use in SCJ to helps programmers with the SCJ memory architecture.

Despite standard Java not being traditionally associated with safety-critical programming, one key feature of SCJ's utility in this area is its foundation in Java. Its familiar syntax and some of its intrinsic features provide a stable base for SCJ.

2.4 *Circus* Introduction

Circus is a state-rich process algebra that combines Z, CSP, guarded commands, and refinement. This allows us to capture the state and behaviour of SCJ programs that we model. Our model also uses features from other members of the *Circus* family. *OhCircus* [18] introduces a notion of object orientation and inheritance, and we use features from *CircusTime* [86] to specify time budgets and deadlines. Figure 2.5 sketches the BNF description of the syntax of *Circus*. Below, we describe the elements of the syntax that are pertinent to the discussion of our formal model. A comprehensive account of *Circus* can be found in [62].

Circus programs, defined in Fig. 2.5 by the syntactic category $\mathsf{Program}$, are formed by a sequence of *Circus* paragraphs. Each *Circus* paragraph ($\mathsf{CircusPar}$) may be either a Z paragraph (the Par category), a channel declaration, a channel set declaration, or a process declaration. The syntactic category N contains the valid Z (and, therefore, *Circus*) identifiers.

Circus programs use bi-directional channels to allow processes to communicate. All of the channels used in a *Circus* program must be declared before use: channel declarations are defined by the CDecl category. If a channel takes any parameters, their types must be declared. Parameter types are drawn from Exp, which is the category of Z expressions. For convenience, channels may be collected into a channel set – defined by elements of the CSExp category. Channel sets allow easy specification of the interface of a process.

Each *Circus* process has a name and a body ($\mathbf{process}\ \mathsf{N} \mathrel{\widehat=} \mathsf{ProcDef}$) and may take parameters. In our model, processes are often parametrised by an identifier of a particular object. The body of a *Circus* process ($\mathbf{begin}\ \mathsf{PPar}^*\ \mathbf{state}\ \mathsf{SchemaExp}\ \mathsf{PPar}^* \bullet \mathsf{Action}\ \mathbf{end}$) is delimited by **begin** and **end**; it may contain a state, which is modelled using a Z schema; and some actions, modelled using a free combination of Z state operations, constructs of a simple imperative language, and CSP constructs (PPar^*). A process's state can be altered

Program	::=	CircusPar*
CircusPar	::=	Par | **channel** CDecl | **channelset** N $\widehat{=}$ CSExp | ProcDecl
CDecl	::=	SimpleCDecl | SimpleCDecl; CDecl
SimpleCDecl	::=	N^+ | N^+ : Exp | $[N^+]N^+$: Exp | ...
CSExp	::=	$\{\!\vert\ \vert\!\}$ | $\{\!\vert N^+ \vert\!\}$ | N | CSExp $\cup$ CSExp | CSExp $\cap$ CSExp
	|	CSExp $\setminus$ CSExp
ProcDecl	::=	**process** N $\widehat{=}$ ProcDef | ...
ProcDef	::=	Decl $\bullet$ ProcDef | Proc ...
Proc	::=	**begin** PPar* **state** SchemaExp PPar* $\bullet$ Action **end** ...
NSExp	::=	$\{\}$ | $\{N^+\}$ | N | NSExp $\cup$ NSExp | NSExp $\cap$ NSExp
	|	NSExp $\setminus$ NSExp
PPar	::=	Par | N $\widehat{=}$ ParAction | **nameset** N $\widehat{=}$ NSExp
ParAction	::=	Action | Decl $\bullet$ ParAction
Action	::=	Command | N | CSPAction | ...
CSPAction	::=	*Stop* | *Chaos* | Pred & Action | Action $\sqcap$ Action
	|	Action $\setminus$ CSExp | $;$ Decl $\bullet$ Action | ...
Comm	::=	N CParameter* | ...
CParameter	::=	?N | ?N : Pred | !Exp | .Exp
Command	::=	$N^+ := Exp^+$ | **if** GActions **fi** | **var** Decl $\bullet$ Action
	|	**val** Decl $\bullet$ Action ...
GActions	::=	Pred $\longrightarrow$ Action | Pred $\longrightarrow$ Action $\Box$ GActions

Figure 2.5: Partial BNF Syntax of *Circus*

by Z schemas or by a direct assignment ($N^+ := Exp^+$, from the Command category).

A *Circus* process always has a main action, at the end of the process after a $\bullet$, that dictates the combination of Z schemas and CSP actions that define the behaviour of the process; these actions may reference other local actions for structuring. Both the state and actions of a *Circus* process are local to that process. This makes *Circus* processes similar to classes in object-oriented programming.

CSP has many operators that are adopted in *Circus*; actions defined using CSP operators all belong to the syntactic category CSPAction. Table 2.3 provides a description of the operators that we use in our model, some of which are omitted in Fig.2.5. Most of them are familiar to users of CSP. We describe them to support the discussion of our model. We note that *Circus* processes can also be combined using most CSP operators.

A simple operator is **Skip**, which terminates and does nothing else. A prefix $c \longrightarrow A$ waits for a communication on the channel c and then proceeds to behave like the action A. Channel parameters can be either an input ($c?x \longrightarrow A$), an output ($c!x \longrightarrow A$), or added to the channel name to indicate a specific communication on that channel ($c.x \longrightarrow A$). This latter form is often used in our models to restrict an action to synchronise on a channel only

Action	Syntax	Description
Skip	**Skip**	A simple operator that terminates
Simple Prefix	$c \longrightarrow A$	Simple synchronisation with no data
Input Prefix	$c?x \longrightarrow A$	Synchronisation that binds a the input value to x
Output Prefix	$c!x \longrightarrow A$	Synchronisation outputting the value of the variable x
Parameter Prefix	$c.x \longrightarrow A$	Synchronisation with some data x
Sequence	$A \; ; \; B$	Executes A then B in sequence
External Choice	$A \,\Box\, B$	Offers a choice between two actions A and B
Conditional	$\textbf{if}\,(x = \mathit{TRUE}) \longrightarrow A$ $\talloblong\,(x = \mathit{FALSE}) \longrightarrow B$ $\textbf{fi}$	Performs A if $x = \mathit{TRUE}$ and B if $x = \mathit{FALSE}$
Interrupt	$A \bigtriangleup c \longrightarrow \textbf{Skip}$	Executes A unless c occurs, which terminates A
Parallelism	$A \llbracket\, ns_a \mid cs \mid ns_b \,\rrbracket B$	Parallelism, synchronising on the channels in c, where A alters the variables in ns_a and B alters the variables in ns_b
Interleaving	$A \llbracket\, ns_a \mid ns_b \,\rrbracket B$	Parallelism with no synchronisation, where A alters the variables in ns_a and B alters the variables in ns_b
Iterated Interleaving	$\Big\vert\Big\vert\Big\vert\; x : S \bullet A(x)$	Interleaving of all actions $A(x)$ where $x \in S$
Recursion	$\mu X \bullet A \; ; \; X$	A process X that executes A then X
Wait	$\textbf{wait}\; t$	Waits for t time units and then terminates
Chaos	**Chaos**	The action that immediately diverges

Table 2.3: Summary of *Circus* operators

if it is parametrised by the identifier of a particular *Circus* process.

A related operator is sequential composition (;), which connects two processes, instead of a channel communication and a process like the prefix operator $\longrightarrow$. Hence $A \, ; \, B$ executes the action A until it terminates and then executes B.

The external choice operator $\Box$ allows an action to offer the choice of two or more different channel communications. Hence $c_1 \longrightarrow A \,\Box\, c_2 \longrightarrow B$ proceeds to A if there is a communication on c_1 or B if there is a communication on c_2. *Circus* also contains a simple conditional statement, as shown in the syntactic category Command in Fig. 2.5. It takes a familiar if…then…else form. Hence $\textbf{if}\,(x = \mathit{TRUE}) \longrightarrow A \;\talloblong\; (x = \mathit{FALSE}) \longrightarrow B \;\textbf{fi}$ performs the action A if $x = \mathit{TRUE}$ and the action B if $x = \mathit{FALSE}$. The interrupt operator $\bigtriangleup$ allows a process to execute unless another process can proceed, in which case the second process interrupts the first. Hence, $A \bigtriangleup c \longrightarrow \textbf{Skip}$ allows the process A to execute, until c occurs (provided that c is not offered by A).

Two actions A and B may be placed in parallel: $A \llbracket ns_a \mid cs \mid ns_b \rrbracket B$, specifies a synchronisation set of channels cs over which both processes must agree to communicate; and name sets (ns_a and ns_b) containing the variables that each side of the parallelism may alter, which must be disjoint to avoid write conflicts. For example, in the execution of $A \llbracket \varnothing \mid \{\!| c_1, c_2 |\!\} \mid \varnothing \rrbracket B$, the actions A and B execute in parallel, but they must agree to communicate on the channels c_1 and c_2 at the same time; further, the use of the empty set ($\varnothing$) indicates that neither A nor B can alter any variables.

A related operator is interleave, which is similar to the parallel operator except that it does not specify a synchronisation set. Hence, $A \llbracket ns_a \mid ns_b \rrbracket B$ allows the processes A and B to execute in parallel with no synchronisations between them. The two name sets, ns_a and ns_b obey the same rules as described above for the parallel operator.

The name sets used by actions in parallel ($A \llbracket ns_a \mid cs \mid ns_b \rrbracket B$) or interleaved actions ($A \llbracket ns_a \mid ns_b \rrbracket B$) control variable access during the parallelism. Essentially, this runs the two actions in parallel, each with its own copy of their process's state. On co-termination, the two copies of the state are merged, according to the variables in the name sets. Where a variable is a member of a name set, its final value is taken from the state of the associated process. If it is in neither name set, then its value remains unchanged. This mechanism does not prevent actions writing to variables not in their name set, but these writes are lost because the version of the variable that is altered is discarded after termination.

In our model, we occasionally need to read a variable on one side of a parallelism or interleaving that is being altered by the other side. To accommodate this we use an internal channel to request the local copy of the variable from the action that is altering it. For example in $A \llbracket varA \mid | get\_a | \mid varB \rrbracket B$, we may have that the action A alters $varA$ and the action B needs to read the new value of $varA$. If B simply uses $varA$, then it reads the version from before the parallelism. Instead, we can use the get_a channel to request the current value of $varA$ from A. In this example, A encapsulates the variable $varA$ in much the same way as a *Circus* process encapsulates the variables in its state component.

Several operators, for example external choice, sequence, parallel, and interleave, can be used in an iterated form. This allows the operator to be applied to all the values in a given set. For example, $\interleave\, x : S \bullet A(x)$ creates an interleaving of all actions $A(x)$ where $x \in S$.

Recursion is defined with the μ operator. Hence, $\mu X \bullet A \,;\, X$ defines a process X that executes the action A then recurses back to X. This is usually combined with an external choice or conditional to provide some condition to break out of the recursion.

Drawn from the *CircusTime* variant of *Circus*, the **wait** operator allows a process to wait for a number of time units. Hence, **wait** t waits for t time units and then terminates. We use this operator to capture time budgets and deadlines.

Finally, the **Chaos** operator is an action that immediately diverges. We use this to model incorrect program states, which allows us to prove that a model does not exhibit these incorrect states by checking for divergence-freedom.

Analysis of *Circus* programs can be performed using ProB [54] and FDR [33]. This requires some skill in manual translation. A tool for automatic translation of *Circus* specifications to CSP$_M$, the input language for FDR, has recently emerged [5]. However, it is a prototype tool that only accepts single-file specifications and potentially has a scalability problem with regard to its translation of state. We return to this issue in Chap. 5.

There are several alternatives to *Circus* as a modelling language. We consider a few below. For example, Event-B [2] is a notation that extends the B method to capture some behavioural information. The behavioural aspects of the specification are captured by event guards, which dictate when an event is enabled.

ABS [10] is an executable specification language that has similar capabilities to *Circus*. Both ABS and *Circus* have an object-oriented model that is similar to Java's and capture concurrency. However, *Circus* contains a refinement calculus that ABS lacks. Refinement is key to the contribution of our model as a target for the *Circus* refinement strategy [19].

$CSP \parallel B$ [82] takes a similar approach to *Circus*. It is a combined notation that uses CSP to capture behaviour, and B to capture state. It aims to utilise the existing tool support for both languages, so the CSP and B [1] parts of a model are each separate and complete. The similarity between $CSP \parallel B$ and *Circus* is underscored by a technique to translate *Circus* specifications in to $CSP \parallel B$ for model checking [91]. This exposes a key benefit of $CSP \parallel B$: tool support. Because the behaviour and state models remain separate, each part of the specification can be model checked by ProB and FDR, respectively. While this is a strength, in terms of analysis, having separate models can be less convenient than *Circus*'s combined approach.

The existing work modelling Java and SCJ in *Circus* means that it is a good choice when approaching formal methods for SCJ. A translation from *Circus* to JCSP, a Java library that implements CSP constructs, has been developed [61]. This technique has been used to model standard Java programs using a translation strategy that transforms *Circus* programs into Java code [31]. This technique is automated in a tool called *JCircus*, which produces a simple GUI for the translated program to facilitate quick prototyping of *Circus* specifications. *JCircus* has recently been adapted to provide an automatic translation from *Circus* to CSP_M [5].

The refinement strategy in [19] provides a technique to relate *Circus* models of different levels of abstraction. A low-level model of SCJ Level 1 already exists [93], which is a target for the refinement strategy. Providing the strategy with a similar target for Level 2 programs is one of the contributions of our work.

2.5 Verification of Safety-Critical Java Programs

Despite being a relatively young language, there are several techniques to verify various properties of SCJ programs. Some of these tools rely on annotations, either custom annotations or the set defined in the SCJ Language Specification. Other tools focus on the source code alone, without annotations, or analyse the Java bytecode. Table 2.4 provides examples of tools available for verifying SCJ programs and the criteria they support.

In the sections below, we discuss the tools and techniques available for verifying SCJ programs, mentioned in Table 2.4. Most of these tools are explicitly aimed at Levels 0 or 1, often ignoring the features of Level 2 entirely. This often makes it difficult to assess their applicability to Level 2 programs. However, we have tried to judge the updates required to make each technique compliant with Level 2 programs.

Criteria	Tool
Worst-Case Execution Time	SafeJML [53] and TetaJ [32]
Worst-Case Memory Consumption	SpideyBC [4]
Memory Safety	privmem [24], JOP Single Nesting Level [67], and *TransMSafe* [57]
Schedulability	TRSL [7] and TetaSARTS [80]
Functional Correctness	SafeJML [53], Java PathExplorer [39], $\mathbf{R}_{SJ}$ [51], and TransCircus [93]

Table 2.4: SCJ Tools and the Safety-Criteria they Address

2.5.1 Worst-Case Execution Time

SafeJML [35] is a specification language for checking functional and timing constrains. It is an extension of the Java Modelling Language (JML) [53], which has minimal memory and timing features. Java source code is annotated with specifications of behaviour and compiled with an extended open-source JML compiler to check the annotations against the code.

JML contains an annotation for specifying the duration of a method, which is effectively its Worst-Case Execution Time (WCET). The duration is measured in JVM cycles, which is not useful in a real-time context. In SafeJML, the duration annotation uses nanoseconds and can check a program's adherence to a WCET specification for methods or blocks.

SafeJML can be used for specifying other properties, as we discuss in Sect. 2.5.5. Further, despite not being explicitly mentioned, it appears to be applicable to all compliance levels. On the other hand, the annotation of programs adds to the overhead of verifying a program.

TetaJ [32] is a tool that translates Java bytecode into a network of timed automata for analysis in the Uppaal model checker. The model of a program is combined with models of the JVM and hardware on which the program will run. This allows model checking of the entire system to estimate the WCET of the program on that JVM and hardware. This analysis is achieved without annotations.

TetaJ seems to be aimed at Level 0 programs. The example application they describe (an implementation of the mine pump control system) runs using a cyclic executive, which is indicative of a Level 0 program. Applicability of TetaJ to other compliance levels is not addressed. The authors mention that the process of modelling a JVM is automated, so integrating a Level 2 compliant JVM (such as the icecap HVM [46]) may be possible. However, it is unclear if the approach to modelling programs is easily adaptable to Levels 1 or 2, which have a different structure to Level 0 programs.

2.5.2 Worst-Case Memory Consumption

Worst-Case Memory Consumption (WCMC) analysis is covered by a tool called SpideyBC, which provides a static analysis of the memory usage of an SCJ program to determine the worst-case consumption [4]. The analysis provides safe upper bound values for the backing store required for memory regions and stack sizes for tasks.

SpideyBC analyses program bytecode to construct a call graph and a control flow graph for each method. The graphs are used to find the most expensive paths through the program in terms of allocated bytes and method invocations that result in frames on the stack, respectively. These paths provide the WCMC values for the backing store and the stack.

SpideyBC restricts the programs it can analyse. All loops and arrays must have explicit bounds, recursion is not supported, and the entire program must be available at compile-time (meaning that dynamic class loading cannot be used, which is trivially satisfied since it is not available in SCJ). The explicit bounds on loops and arrays are specified using annotations in the source code, which we assume to be custom annotations designed by the authors.

To our knowledge, this is the only technique for calculating WCMC for SCJ programs. The authors test their technique on three small example applications and manually compare the most expensive paths suggested by SpideyBC with other possible paths. They hope that their technique provides the impetus for others by providing a useful benchmark.

SpideyBC is explicitly aimed at Levels 0 and 1. The underlying technique (discovering the most expensive execution paths) is compliance-level agnostic. But, to ensure that is it amenable to Level 2, SpideyBC needs updating to accept programs with multiple active missions and managed threads so that it can build accurate graphs.

2.5.3 Memory Safety

Early versions of the SCJ Language Specification contain a set of annotations for ensuring the safety of memory references. The memory areas in which objects reside are tagged within the program, and the checker tool analyses these annotations to prove the memory safety of a program [77]. Problematically, the implementation of the checker tool revealed that if a class is required in several memory areas, then it must be duplicated. Further, in the current draft (v 0.100) they have been moved to an appendix because they ‘were not ready for standardization’ [79]. This means that SCJ implementations need not include the annotations, so their presence cannot be depended upon as a general technique for analysing SCJ memory safety.

A hardware-based technique for checking memory safety of Level 0 or 1 programs has been implemented on the Java Optimised Processor (JOP) [67]. This technique assigns a nesting level to each of SCJ’s memory areas: immortal memory is level 0, the top-level mission memory is level 1, and so on.

When assignments are made, the nesting level of the variable is compared to the nesting level of the reference being assigned. If the variable is static, then its nesting level must be 0, because all static variables reside in immortal memory. For other variables, the reference must be at a lower (more deeply nested) nesting level than the variable. These checks ensure that memory references only point to longer-lived memory areas.

Using a single nesting level only caters to the simpler memory hierarchy of Levels 0 and 1, where the the memory areas can only be nested linearly. In Level 2, the memory hierarchy can grow as a tree, because of the nested memory areas. However, this technique can be used to check parts of a Level 2 program where the relationship between the memory areas is linear. For example, an outer nested mission memory cannot reference an inner mission memory.

The privmem tool [24] performs a static analysis of Java bytecode to prove that a program does not attempt to violate the SCJ memory rules, without the need for annotations. If the input program does violate the memory rules, then privmem provide a counter-example. The privmem analysis constructs an over-approximation of potential memory rule violations by recording all the possible references from variables to objects. This is achieved by tracking the current allocation context throughout the program, following the methods that alter it, and updating the allocation contexts of new objects.

However, privmem does not track allocations made using the methods that allow allocations directly into other scopes: `newArray()`, `newArrayInArea()`, and `newInstance()`. Further, it allows the SCJ infrastructure to make temporary memory rule violations: if such violations occur, they are ignored. This diverts from the SCJ language specification, without making a case for why the infrastructure needs to violate the memory rules.

Privmem struggles to track the usage of mission memory. The mission's `initialize()` method is used to track the initial entry to the mission memory, but there is no API method that the tool can use to track the re-entry to mission memory before the schedulable objects are activated. Also, because privmem performs its analysis on Java bytecode, tracing the location of a counter-example back to the source code can be more difficult than with source code analysis. Finally, the example applications used to evaluate privmem all appear to be Level 0, since they only mention periodic event handlers.

While the technique of tracking where the program allocates memory is compliance-level agnostic, the privmem tool needs updating to cater for Level 2 features, such as nested mission sequencers and managed threads. This seems possible, but one challenge is how the technique would deal with multiple active mission memory areas, particularly as it struggles to track the usage of a single active mission memory.

TransMSafe [57] automatically analyses the memory safety of SCJ programs. It translates the source code of an SCJ program into *SCJ-mSafe*, which is a novel abstract language for describing the memory usage in SCJ programs. The translation is formalised in Z, which paves the way to a proof of the soundness of the technique.

To check the *SCJ-mSafe* program, the tool builds an environment of reference variables mapped to their reference contexts. A reference context is any of the memory areas available in the program, plus a context for primitive values. Building this environment covers all execution paths, so it captures all possible allocations that the program might make. This environment is checked to ensure it adheres to the SCJ memory rules.

TransMSafe over-estimates the allocations that can occur in the program, so it can raise false negatives. Further, it is explicitly aimed at checking the memory safety of Level 1 programs, so it is not directly applicable to Level 2 programs. However, this technique could be extended to cover Level 2. Currently, an *SCJ-mSafe* program only allows one mission sequencer, where Level 2 programs may have many. Further, the technique does not address managed threads because they are only available at Level 2. Finally, the analysis treats missions individually, since they are sequential, in Level 1 programs. To accept Level 2 programs, *TransMSafe* needs to either analyse each of the top-level mission sequencer's missions along side all of the missions from other mission sequencers or be adapted to take execution order of missions into account.

2.5.4 Schedulability

The Time and Resource Specification Language (TRSL) [7] provides an approach for checking the schedulability of SCJ programs. TRSL specifications of execution time and locking behaviour are included in the program as annotations. A specification is an abstract trace, which is composed of a sequence of blocks. Each block can be: skip, which takes no time; a time interval; a trace; a usage block, which describes a critical section protected by locks; a repeat block, which describes the repetition of a block a number of times; or a select block, which is a non-deterministic choice between a set of traces.

To check the schedulability of the program, its traces are translated in to a network of timed automata, which is checked using Uppaal. Schedulability of the modelled program is indicated by deadlock freedom. The technique also allows a program to be checked to ensure that it implements its specification.

A similar approach, TetaSARTS [80], merges many ideas, including the TetaJ WCET technique discussed in Sect. 2.5.1. TetaSARTS generates a network of timed automata for the SCJ input program and analyses it using Uppaal, to perform schedulability analysis.

TetaSARTS differs from the TRSL approach in that it generates the model from Java bytecode, however it still requires annotations for loop bounds, like the TetaJ tool. Schedulability of the model is, again, shown by deadlock freedom of the model.

The example applications used to evaluate both of these techniques make no mention of Level 2 features, so we assume that they are aimed at Level 0 or 1. It appears that they are both applicable to Level 2 programs. But, their respective tools need to be updated to accept Level 2 programs for analysis by, for example, ensuring that they are expecting managed threads and multiple mission sequencers. Further, the techniques need to be able to handle the ability of Level 2 programs to allocate schedulable objects to multiple processors, which is not present at the lower compliance levels. Finally, both techniques require annotations, which increases the programming overhead. TetaJ requires the annotation of all the program's methods; TetaSARTS only needs annotations on loop bounds, which is less onerous.

2.5.5 Functional Correctness

The SCJ Language Specification contains a set of annotations that can be used to specify that a class or class member may only be used in a particular compliance level, that a method may only be used during a particular phase, that a method may self suspend, and the memory areas that a method may allocate in. The SCJ annotation checker [77], discussed in Sect. 2.5.3, can be used to statically check SCJ's behavioural annotations.

The checker tool was constructed to be compliant with an earlier version of the SCJ Language Specification, and some of the annotations have changed in the current version of the specification [79]. First, the `@SCJAllowed` annotation defines that code is visible to other code at a particular compliance level or infrastructure code (using the `SUPPORT` argument). However, the checker tool allows further values of the annotation's argument, which have been subsequently removed. Three separate annotations in the current version of the specification (`@SCJMayAllocate`, `@SCJMaySelfSuspend`, and `@SCJPhase`) are arguments to one annotation (`@SCJRestrict`) in the version with which the checker tool is compliant. Finally, as previously mentioned, the memory safety annotations that the checker tool accepts are not included in the main SCJ Language Specification because they are not yet ready for standardisation.

The annotation checker statically checks the conformance of the program with its annotations. However, the overhead of annotating an entire program, particularly with SCJ's novel programming and memory paradigm, potentially outweighs the usefulness of being able to automatically check them. While the annotations cover useful metadata about the program, they are inflexible and cannot be extended to check for custom program properties.

SafeJML [35] has also already been mentioned; it extends JML and uses custom Java annotations to specify functional and timing constraints of SCJ programs. The input program's source code, complete with annotations, is compiled by the SafeJML compiler to check that the program complies with its annotations. SafeJML reports violations of statement annotations by throwing an error.

SafeJML can be used to annotate a method or block with its duration, which (as we discuss in Sect. 2.5.1) can be used to specify and check WCETs. Other annotations include specifications of: maximum loop iterations, maximum executions of condition-guarded blocks, and execution paths. Further, SafeJML handles subtype polymorphism. It uses 'model methods' (side-effect free methods at are only intended for specification purposes) to allow the specification of properties that can vary with the runtime type of an object. Again, the downside of this technique is that the entire program must be annotated with behavioural specifications.

Java PathExplorer [39] (JPE) is a tool that provides online monitoring of Java execution. This is achieved by instrumenting the Java bytecode of a program to emit events to an observer module, which is used to verify the execution of the program. This can either be high-level verification, where the executing program is compared to user-provided requirements specifications, or lower-level error detection, which is usually concerned with concurrency related problems like race-conditions and deadlocks.

JPE accepts high-level specifications in Maude rewriting logic [20], which can be checked using the Maude rewriting engine or translated to, and checked in, Java. JPE is intended to aid testing, particularly the integration of formal methods with testing to avoid the problems with ad-hoc testing and the complexity of theorem proving. It can also be used to provide run-time verification and to influence program behaviour if its requirements are violated.

Java PathFinder (JPF) is a model checker for Java. The first version of JPF [38] translates Java programs into Promela models, which are checked for deadlock freedom and adherence to assertions (which are translated from any `assert` statements in the input program). This technique requires the input program to have a finite and tractable state space.

While the first version of JPF is a useful approach to model checking Java programs, it suffers from two drawbacks. Firstly, every Java expression has to be mirrored in Promela. This is not always possible: for example, Promela does not support floating point numbers. Secondly, the translation requires access to the source code. Again, this is not always possible: for example, programs may use libraries for which only the Java bytecode is available [85].

The updated version of JPF in [85] departs from the idea of translating the input program and, instead, uses a novel custom model checker, which consists of a custom JVM and a search component to guide executions. JPF executes input programs to ensure that every possible execution path is explored, from each choice point or instance of nondeterminism. A choice point occurs when the program takes input values or a thread is chosen for execution.

The $\mathbf{R}_{SJ}$ tool [51] extends the JPF to allow it to accept SCJ programs. This is achieved by providing a novel scheduling algorithm to execute in an SCJ compliant way. $\mathbf{R}_{SJ}$ can discover memory access errors, race conditions, priority ceiling emulation protocol violations, and other application-specific run-time errors, like dereferencing a null pointer, invalid arguments to library code, array bound violations, division by zero, and failed assertions.

$\mathbf{R}_{SJ}$ is explicitly aimed at Levels 0 and 1, but without aperiodic event handlers because their arbitrary release times cause state explosion. It is, therefore, not directly suitable for Level 2 programs. Since aperiodic event handlers lead to intractable models, it is unlikely that $\mathbf{R}_{SJ}$ would scale well to the more complex concurrency available at Level 2.

As previously mentioned, the model of SCJ programs in [93] provides directly represents the expressions in the program, but at a more abstract level. It captures the SCJ paradigm and the programs separately in *Circus*. The paradigm model is generic and reusable; programs are translated by an automatic tool called TransCircus. Combining the two separate *Circus* components produces a model that exhibits the behaviour of the program and is amenable to model checking, via a translation of the model into CSP_M for input to FDR.

The aim of the work in [93] is not solely to provide a model checking technique for SCJ programs. It it also an important part of a *Circus* refinement strategy [19] that translates abstract specifications of behaviour into concrete models of SCJ programs. This provides a correct-by-construction technique for SCJ.

The model and translation provided by [93] are explicitly aimed at Level 1, and do not cover some features that were either considered to be too complex or were not part of the language specification at the time of modelling. We take this work as inspiration in modelling SCJ Level 2. We describe our approach in full, in Chap. 4. Hence, the work presented in this thesis addresses the functional correctness of SCJ Level 2 programs.

2.6 Summary

SCJ is a Java-based programming language for systems that must be certified. It restricts the region-based memory and concurrency abstractions of the RTSJ, and provides a novel hierarchical programming paradigm. Despite SCJ's restrictions, its programs, especially at Level 2, can be very complex. Level 2's unique features can capture safety-critical use cases that are not possible at Levels 0 or 1. Importantly for our work, however, SCJ programs are amenable to modelling, as shown by the *Circus* models of different aspects of SCJ [16, 93, 59].

Circus and its extensions provides features for modelling state, behaviour, objects, and time. This makes it useful for modelling languages like SCJ, which are object-orientated and real-time. *Circus* produces readable specifications, but tool support is a weakness. In particular, model checking *Circus* requires translation to CSP_M for FDR. This can be achieved manually, which requires some skill, or with a translation tool [5], which has limitations that we discuss further in Chap. 5.

Despite being a young language, there are already several verification techniques aimed at SCJ. However, the majority of techniques are not aimed at Level 2. There are some techniques, such as SafeJML [35], that are applicable to all SCJ compliance levels. However, Level 2 has not been the direct focus of verification of WCET, WCMC, memory safety, schedulability, or functional correctness.

Despite its restricted structure, Level 2 programs can become very complex. This has likely lead to the lack of verification techniques aimed at Level 2. Most of the techniques available for SCJ seem to be amenable to extension for Level 2, as long as their tool support is updated to cater to Level 2's features. The *Circus* model of the Level 1 paradigm strikes a useful balance between abstraction and close correspondence with programs, so we use this as the foundation for our approach to address the functional correctness of Level 2 programs.

Chapter 3

Applications and Evaluation of SCJ Level 2

This chapter discusses in detail the utility of the unique features of SCJ Level 2 and presents use cases that justify the availability of these features. This is the first such examination of the features and utility of SCJ Level 2. This chapter is based upon the work published in [89] and [56]. Section 3.1 describes Level 2's features, which frames the discussion in this chapter. Section 3.2 describes two programming patterns that require Level 2's unique features. Section 3.3 shows the utility of managed threads, in providing extended release patterns, when combined with suspension, and better encapsulation of state. In Sect. 3.4 we describe specific challenges arising from Level 2's unique features identified following the studies reported in Sects. 3.2 and 3.3. To meet these challenges, we propose changes to the SCJ specification, one of which has already been accepted by the standardisation group. Finally, in Sect. 3.5, we summarise the utility and challenges of SCJ Level 2.

3.1 SCJ Level 2: Unique Features

The SCJ Language Specification is clear on what constitutes a Level 2 program but not why it should be used. From the features of SCJ Levels 0 and 1, described in Sect. 2.1, their utility is clear. Level 0 programs are periodic and executed by a cyclic executive. So a program's required scheduling behaviour is a primary indicator of whether or not Level 0 should be used. However, both Levels 1 and 2 use concurrency and fixed-priority scheduling, so this cannot be used as an indicator to choose between them.

To understand the purpose of Level 2, we examine the application-level programming requirements for which its functionality is necessary. These requirements are not included in the rationale for the compliance levels in the SCJ Language Specification. We broadly classify Level 2's unique features into four groups:

- Nested `MissionSequencers`,
- `ManagedThreads`,
- Suspension, and
- Global Scheduling over Multiple Processors.

Since our modelling approach is agnostic to the number of processors in use, we ignore the issue of global scheduling.

The most prominent of Level 2's unique features, in terms of structure, is the possibility of multiple active missions. This is achieved by allowing *nested* mission sequencers, which are started by a mission. Each nested mission sequencer has a separate sequence of missions. By contrast, Level 0 or 1 programs only have one mission sequencer, which is started by the safelet and has a single sequence of missions.

Managed Threads have a simpler release pattern than that of SCJ's event handlers: they are released immediately and run to completion, with no deadline. Their release behaviour is captured in their `run()` method; its default memory area is active for the duration of this method. The length of a managed thread's release can be as long as required; a loop may be used within its `run()` method, or the thread may suspend. The practical outcome of this is that the memory area of a managed thread can be kept active for as long as needed. Finally, SCJ Level 2 programs are allowed to use the the `Object.wait()` and `Object.notify()` family of methods from standard Java to perform suspension-based waiting

3.2 Nested Mission Sequencers

The ability to construct applications composed of nested mission sequencers is, perhaps, the most important aspect to be considered when choosing between Levels 0 or 1 and Level 2. The benefit of nested mission sequencers is primarily structural. They provide better control over the schedulables in the program and the way that they execute.

In this section we identify two software architecture patterns that require nested mission sequencers, and sketch an example application for each. We call these two patterns *Multiple-Mode Applications* and *Independently Developed Subsystems*. Since both of these pattens are examples of the utility of nested mission sequencers in SCJ Level 2 programs, we pick an implementations of the Multiple-Mode Applications pattern (described in Sect. 3.2.1) to help evaluate our translation in Sect. 5.4.

3.2.1 Multiple-Mode Applications

This pattern captures the typical architecture of systems that operate in multiple modes. There may be *persistent* tasks, which operate in all modes, and *mode-specific* tasks, which only operate in one mode. Schedulability analysis techniques can be used to guarantee the timing properties in the steady-state situations of execution in each mode. Analysis techniques also exist for handling the transitions between modes, but only on a single processor [81, 66].

This pattern can be programmed at Level 1, but Level 2 captures the requirements of the application better. Level 1 lacks nested mission sequencers, so persistent tasks must be duplicated in each mode. This duplication means that their execution is interrupted by mode changes. If the persistent tasks have state, it must be stored in a higher-level memory area (such as the immortal memory area) to prevent it from being lost during mode changes.

Level 2 allows persistent tasks to run uninterrupted by mode changes and, as we discuss in Sect. 3.3.3, retain their state locally. Further, Level 2 allows a multiple-mode system to be included as a component in a more complex program.

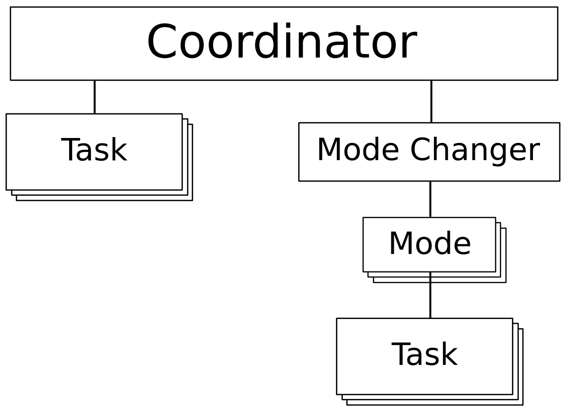


Figure 3.1: Multiple-Mode Applications Pattern

Architecture Components

The components that characterise this pattern are shown in Fig. 3.1. A coordinator controls a mode changer and any persistent tasks. Persistent tasks are required to operate during all modes. A mode changer encapsulates several modes, and each mode encapsulates mode-specific tasks. A mode changer can only have one active mode at a time. Mode changes are typically requested by tasks from the currently active mode.

In SCJ, a mode changer can be conveniently implemented as a mission sequencer, each mode as a mission, and each task as a schedulable. The coordinator component also has a natural correspondence with a mission, often the main mission, which registers the persistent tasks and the mode changer, and controls their operation.

Example Application

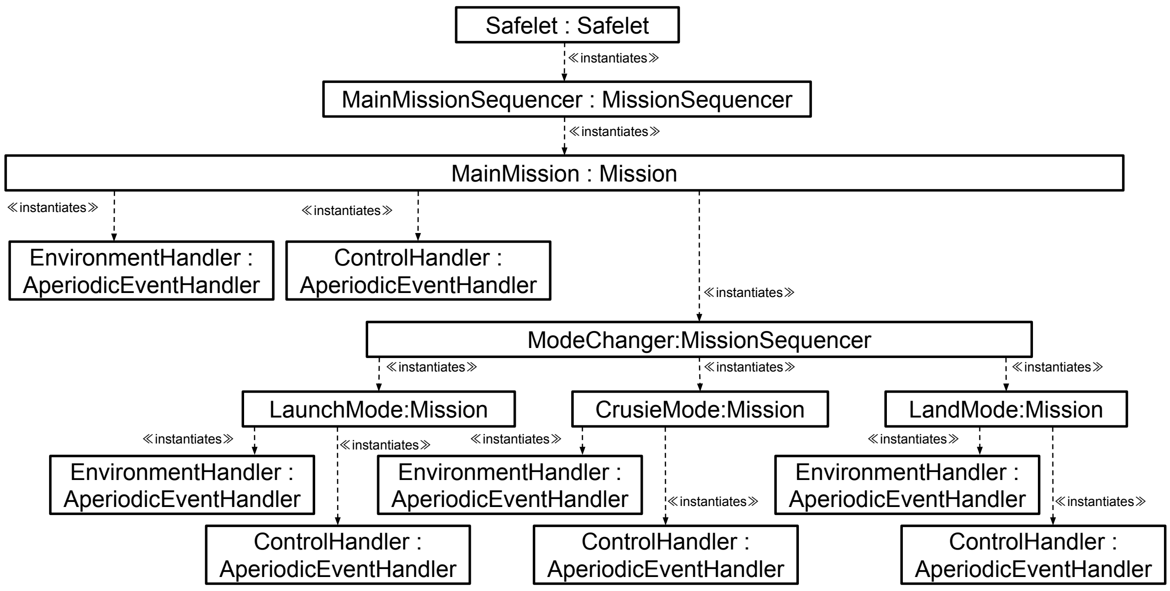
An example application that uses this pattern is a simplified Space Shuttle[1], as shown in Fig. 3.2. Each of its three modes is associated with a phase of operation, and has several mode-specific schedulables. Fig. 3.2 only shows two mode-specific schedulables per mode and two persistent schedulables (`EnvironmentMonitor` and `ControlHandler`), for brevity.

At the start of the execution phase of the main mission, the persistent schedulables begin executing alongside those from the `LaunchMode`. Once the launch is complete, the `LaunchMode` is requested to terminate. When it has, the mode changer loads the `CruiseMode`, and its schedulables execute alongside the persistent schedulables, which remain active during the mode change. When the craft needs to land, the `CruiseMode` is terminate and the `LandMode` is loaded, which runs until the craft lands and the program can terminate.

Adequacy of SCJ Support

Using missions to support individual modes of operation and mission sequencers to control the mode-changes has two main advantages. The first is that encapsulating each mode in a mission enhances the modularisation of the program and the traceability of its structure to its architectural design. This is especially important when each mode is a significant software component in its own right, as is the case in our example.

[1]The code for this example can be found at `http://www.cs.york.ac.uk/circus/hijac/case.html`



Safelet : Safelet
≪instantiates≫
MainMissionSequencer : MissionSequencer
MainMission : Mission
EnvironmentHandler : AperiodicEventHandler
ControlHandler : AperiodicEventHandler
ModeChanger:MissionSequencer
LaunchMode:Mission
CrusieMode:Mission
LandMode:Mission

Figure 3.2: Space Shuttle with Multiple Modes

The second advantage is that SCJ supports a well-defined process for mission termination, where schedulables can complete their current release before the mission terminates. Usefully, this protocol supports mode change requests when they are *planned* events. Planned mode changes occur at well defined points in a system's operation. By contrast, *unplanned* mode changes usually occur as a result of error conditions being detected. Such errors may be anticipated, but the time of their occurrence can not be predicted. Hence the time at which a mode change is required cannot be predicted; they are unplanned.

Using the multiple-mode applications pattern in SCJ raises some timing issues. First, in order to execute a new mode, it is necessary to create all the new objects that are to reside in the mission memory, during the initialization phase of the mission (mode). Hence, for unplanned mode changes or applications that require fast and predictable planned changes, there may be some efficiency or latency concerns.

Further, there is no automatic single release time for all of a program's schedulables. The persistent schedulables start before those in the first mode. A single start time can be created manually for event handlers, using a periodic or one-shot event handler's start time offset (aperiodic event handlers are only released upon request). However, managed threads are released immediately. Controlling their start time would require manually programming a release mechanism, as discussed in Sect. 3.3.

Multiple-mode applications can be analysed for timing properties by treating the mission sequencer implementing the mode changer as an aperiodic task. Its minimum inter-arrival time is equal to the minimum time between mode change requests. Its deadline represents any time constraints on the mode change operation.

An SCJ mission sequencer is a subclass of event handler so it only has a priority (for scheduling) and storage parameters (for the size of its memory area); it does not have any release parameters. To perform schedulability analysis on mission sequencers, their release properties must be captured outside the program. We discuss this concern further in Sect. 3.4.

Finally, SCJ does not support hierarchical scheduling, which makes it challenging to support compositional time analysis of the application. SCJ schedules persistent and mode-

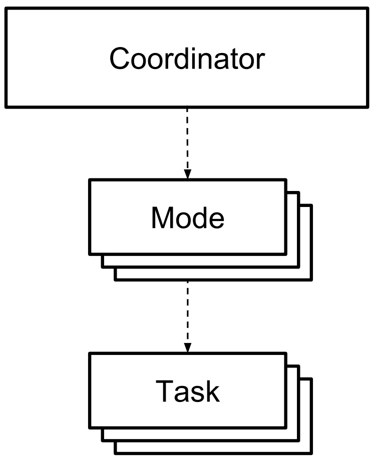


Figure 3.3: Independently-Developed Subsystem Pattern

specific schedulables in competition. Hence, the whole application must be analysed in each mode along with each mode transition. We return to this issue in Sect.3.4.3.

3.2.2 Independently Developed Subsystems

Complex systems can be composed of one or more subsystems, which encapsulate and control similar behaviour. These subsystems may be developed independently of each other and of the main program. This pattern considers these subsystems to be a program in their own right, each performing particular related behaviours. Nested mission sequencers are the key to supporting this approach to constructing systems in SCJ.

Level 1 can capture the functionality of this programming pattern. However, at Level 1, all the behaviours would have controlled by one mission, effectively flattening all the subsystems into one. This complicates the independent development of a subsystem. Level 2 provides better encapsulation and control of individual subsystems. For example subsystems can be terminated or restarted independently of the main program and other subsystems. Using this programming pattern at Level 2 provides a uniform method of plugging an independently developed subsystem into a program.

Architecture Components

The architecture that characterises this pattern is shown in Fig. 3.3. Each subsystem contains several tasks that perform related behaviours; they may also contain other subsystems. Each subsystem may be developed independently and then integrated into the program. A coordinator component controls and integrates the subsystems.

In SCJ, each subsystem can be implemented using: a mission sequencer, which allows the integration of the subsystem into the program, and a single mission that manages the tasks within that subsystem. Each task can then be implemented by an appropriate schedulable. The coordinator component corresponds naturally to a mission, often the main mission, that registers the mission sequencers controlling each subsystem.

Example Application

A good example of this pattern is the railway system described by Hunt and Nilsen [43]:

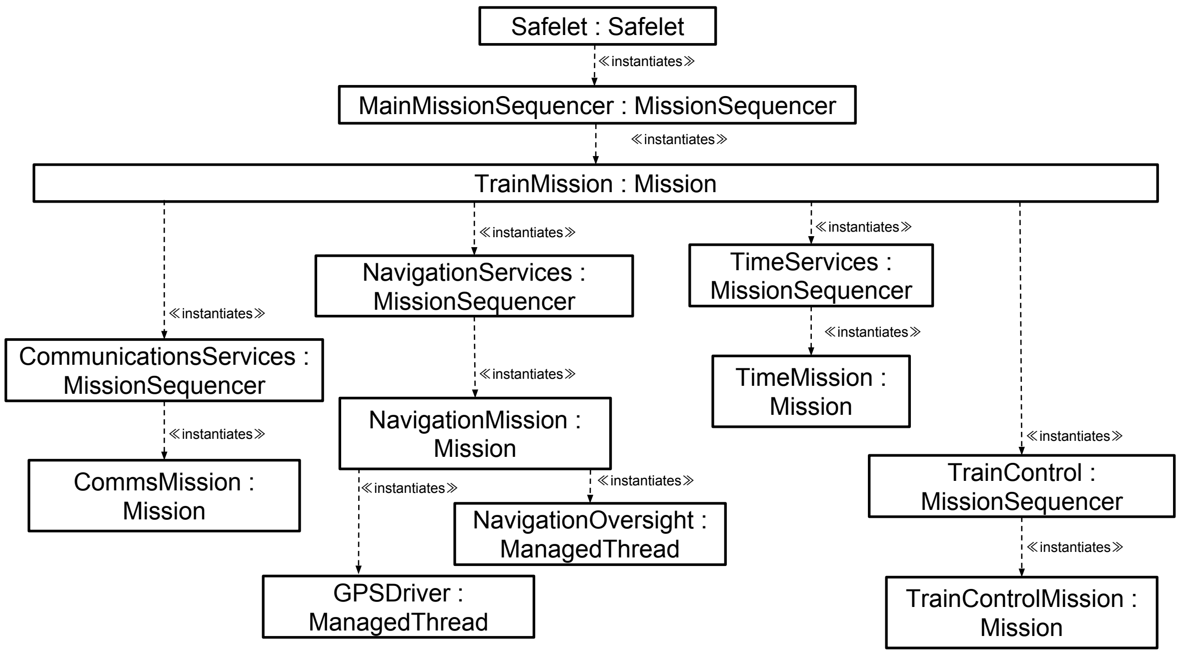

Safelet : Safelet
≪instantiates≫
MainMissionSequencer : MissionSequencer
≪instantiates≫
TrainMission : Mission
≪instantiates≫
CommunicationsServices : MissionSequencer
≪instantiates≫
CommsMission : Mission
≪instantiates≫
NavigationServices : MissionSequencer
≪instantiates≫
NavigationMission : Mission
≪instantiates≫
NavigationOversight : ManagedThread
≪instantiates≫
GPSDriver : ManagedThread
≪instantiates≫
TimeServices : MissionSequencer
≪instantiates≫
TimeMission : Mission
≪instantiates≫
TrainControl : MissionSequencer
≪instantiates≫
TrainControlMission : Mission

Figure 3.4: Railway System with Multiple Subsystems

> 'Collision avoidance in rail systems is a representative safety-critical application. A common approach to the challenge of avoiding train system collisions divides all tracks into independently governed segments. A central rail traffic control system takes responsibility for authorizing particular trains to occupy particular rail segments at a given time. Each train is individually responsible for honouring the train segment authorizations that are granted to it. Note that rail segment control addresses multiple competing concerns. On the one hand, there is a desire to optimize utilization of railway resources. This argues for high speeds and unencumbered access. On the other hand, there is a need to assure the safety of rail transport. This motivates lower speeds, larger safety buffers between travelling trains, and more conservative sharing of rail segments.'

Their example considers the structure of the on-board software (illustrated in Fig. 3.4), which supports the following requirements:

- maintain reliable and secure communication with the central rail traffic control authority, provided by the *CommunicationServices* subsystem;
- monitor progress of the train along its assigned route, provided by the *NavigationServices* subsystem;
- control the train's speed in accordance with scheduled station stops, rail segment authorizations, local speed limit considerations, and fuel efficiency objectives, provided by the *TrainControl* subsystem; and,
- maintain global time, provided by the *TimeServices* subsystem.

In the implementation described in [43], each of these subsystems is implemented as a nested mission sequencer registered to the main mission (`TrainMission`), and each subsystem controls a single mission that registers the subsystem-specific schedulables. There are multiple tiers of nested mission sequencers within the subsystems. Each tier represents further subsystems that can be developed independently. For brevity, Fig. 3.4 only shows that two subsystem-specific schedulables of the Navigation Services subsystem and omits the deeper tiers of nested mission sequencers.

Adequacy of SCJ Support

Although the encapsulation provided by missions is ideal for structuring subsystems, there are issues that need to be addressed when adopting this approach. The first is that in order to compose a system from many subsystems, each of the missions that controls a subsystem must be controlled by its own mission sequencer. This is natural if each mission has multiple modes of operation, but can become cumbersome otherwise.

Secondly, as already mentioned in Sect. 3.2.1, when a system is composed of subsystems, there is no automatic common release time for all the schedulables. If required, this has to be programmed explicitly. For multiple nested mission sequencers, this can become awkward because the system start time needs to be passed down to all schedulables in the program.

Whilst the above limitations can be seen as minor, the third is more significant. Namely, it is difficult to decompose timing constraints when subsystems are independently developed, because neither SCJ nor the RTSJ directly support hierarchical scheduling. The RTSJ supports processing groups, which allow several schedulables to share a CPU budget, but these are too general and difficult to use in a multiprocessor environment [13, 87]. Hierarchical scheduling techniques for single processor and partitioned multiprocessor systems are well established [26] and techniques are beginning to emerge for globally scheduled multiprocessor systems [11, 27]. The lack of this facility in SCJ severely limits its support for the timing analysis of applications using this pattern. We return to this issue in Sect. 3.4.3.

3.3 Managed Threads and Suspension

The managed threads and suspension available in SCJ Level 2 applications provide several benefits. Their utilities often complement each other, so we discuss them both here. We use an application that combines managed threads and suspension (described in Sect. 2.1.2) to help evaluate our translation, presented in Sect. 5.4.

SCJ provides four release patterns: periodic, aperiodic, one-shot, and the simple run-to-completion exhibited by managed threads (which is only available at Level 2). The simple run-to-completion release pattern of managed threads can be used to support background activities that run as fast as possible when they have access to the processor. For example, a logging task that processes data from application logs whenever it is scheduled. There is no notion of release events for these activities, other than their initial release.

At Level 1, this could be achieved with an aperiodic event handler that is only released once, when the program starts, or by a one-shot event handler that has no start time offset and is released immediately. However, this is a misuse of these tasks, which embody a particular release pattern. Although there is no negative consequence for this misuse, managed threads are a cleaner abstraction to support this sort of activity.

Combining managed threads and suspension allows the programming of release patterns not provided by SCJ, which we discuss in Sect. 3.3.1. Suspension itself is a useful feature, and we discuss its utility in Sect. 3.3.2. Finally, we show how managed threads allow better encapsulation of the state of schedulables than can be programmed at Level 1, in Sect. 3.3.3.

3.3.1 Extended Release Patterns

As previously mentioned, SCJ provides four release patterns, each implemented as a different schedulable. By combining managed threads and suspension, one can program extended release patterns. In SCJ, a periodic event handler is released either immediately or after an absolute or relative delay from when it is started. Here we consider the example of a periodic task that is released by a software notification. Such a release pattern supports a mechanical system that requires periodic control, but is started by an aperiodic button press.

Using Level 2's suspension features to allow a periodic event handler to wait for a notification, is not sufficient. Deadline monitoring of event handlers begins when the handler is first released and they cannot be dynamically changed. This means that it is not possible to set an initial deadline and then update it after the notification has occurred.

Managed threads can be extended to provide this release pattern. We consider, for example, the class shown in Fig. 3.5, which extends `ManagedThread` to provide a periodic thread that is released by a method call. The `run()` method (lines 24-34) is final and waits for the initial release before calling the `work()` method periodically. The abstract `work()` method declared on line 36 must be overridden to provide the functionality to be called each period. The `firstRelease()` method (lines 38-44) is called during the mission to release the periodic activity. This example illustrates the additional flexibility that is available at Level 2; the periodic thread in Fig. 3.5 cannot be programmed using SCJ Level 1.

This release pattern can be extended to program the thruster control system described by Wellings [88, Page 235]. Here, an astronaut activates the thruster and supplies a duration for the engine 'burn'. The engine requires periodic control to avoid the mechanical drift of its valves. This requires an activity that is released by an event, executes periodically for a certain duration (determined either by time itself or by another event), and then waits to be started again. This can be programmed in SCJ by adapting the periodic thread in Fig. 3.5 to call `waitFirstRelease()` after a certain time offset and wait to be released again.

3.3.2 Suspension-based Waiting

Level 2 programs may use suspension-based waiting, which allows them to capture many use cases that Levels 0 and 1 cannot. A simple example is device drivers, which often busy-wait for input or output to complete because the expected delay is small and context switching away from the driver is inefficient.

There are ways to integrate this delay into a driver's scheduling (see [12, Section 14.6]) or allowing the driver to delay when it has no other activity to perform. But, when the delay is relatively long, it is necessary to allow the system to schedule some alternative activities.

Since it is not possible to have a suspension-based delay in Level 1 programs, this requirement can only be implemented at Level 2. Another example of where suspension-based

```
public abstract class PeriodicThread extends ManagedThread {

  private final int period;
  private final int deadline;
  private AbsoluteTime nextRelease; //the next release time of this thread
  private AbsoluteTime nextDeadline; // the next deadline of this thread
  private DeadlineMissHandler deadlineMissDetection;
  private Mission myMission; //this thread's controlling mission
  private boolean hadFirstRelease = false;

  public PeriodicThread(int period, int deadline, PriorityParameters priority) {
      super(priority);
      this.period = period;
      this.deadline = deadline;
      nextRelease = new AbsoluteTime();
      nextDeadline = new AbsoluteTime();
      deadlineMissDetection = new DeadlineMissHandler();
      myMission = Mission.getCurrentMission();
  }

  private synchronized boolean waitFirstRelease() {
    while(!hadFirstRelease){
      try {
        wait();
      } catch(InterruptedException ie) {
        // mission is to be terminated
        return false;
      }
    }
    return true;
  }

  public final void run() {
    if (waitFirstRelease()) {
      while(!myMission.terminationPending()) {
        nextRelease.add(period,0);
        work();
        nextDeadline.add(period,0);
        deadlineMissDetection.scheduleNextReleaseTime(nextDeadline);
        Services.delay(nextRelease); // waitForNextPeriod
      }
    }
  }

  protected abstract void work(); // override to provide thread's behaviour

  public synchronized void firstRelease() {
    hadFirstRelease = true; notify();
    nextRelease = Clock.getRealtimeClock().getTime(nextRelease);
    nextDeadline.set(nextRelease.getMilliseconds() + deadline);
    deadlineMissDetection.scheduleNextReleaseTime(nextDeadline);
  }
}
```

Figure 3.5: A Periodic Task Released by Software Notification

```
public class Consumer extends ManagedThread {
  private final PCMission pcMission;
  private final Buffer buffer;

  ...

  public void run() {
    while (!pcMission.terminationPending()) {
    try {
      result = buffer.read();
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
      Console.println("Consumer Read " + result + " from Buffer");
    }
  }
}
```

Figure 3.6: Consumer thread

waiting is needed is producer-consumer systems.

Producer-Consumer Systems

Producer-consumer systems involve tasks generating data (the producers) that is to be processed by other tasks (the consumers). The data is written to and read from a shared bounded buffer. Here, the data often comes in bursts. A common solution to producer-consumer problems uses suspension: the producers suspend when the buffer is full and the consumers suspend when the buffer is empty.

Clearly these requirements cannot be met at Level 1, as it does not allow suspension. It initially seems that producers and consumers could be implemented using aperiodic event handlers, where a handler is released each time the buffer is full or empty – depending on if it is a producer or a consumer. However, SCJ does not allow a queue of outstanding release events for aperiodic event handlers, so they are not appropriate. Level 2 enables this release pattern to be programmed using managed threads and suspension.

As an example, we consider a simple producer-consumer system with one producer, one consumer, and a `Buffer` object to which they share access. Figure 3.6 shows the consumer task, which extends `ManagedThread`. It holds a reference to its controlling mission and the `Buffer` object – which is held in mission memory. In this simple example, the `Buffer`, shown in Fig. 3.7, holds a one-place buffer.

The consumer's `run()` method calls the synchronised `buffer.read()` method, which encapsulates the behaviour of reading from the buffer (including the potential suspension). The `read()` method first checks the status of the buffer, using the `bufferEmpty()` method. The call to `bufferEmpty()` is in a while loop to guard against spurious wake ups. If the buffer is empty, then it calls `this.wait()`, which suspends the consumer thread on the buffer. If the buffer is not empty or the consumer has resumed, then the consumer: reads and clears the buffer, then calls `this.notify()` to wake the producer. The producer thread has similar behaviour, except it will wait when the buffer is full.

```
public class Buffer {

  private volatile int buffer;

  public boolean bufferEmpty() {
    return buffer == 0;
  }

  ...

  public synchronized int read() throws InterruptedException {
    while(bufferEmpty()) {
      this.wait();
    }
    // Read buffer
    int out = buffer;
    buffer = 0;
    this.notify();

    return out;
  }
}
```

Figure 3.7: Buffer Object

3.3.3 Encapsulation of State Information

Memory usage is another differentiating factor between managed threads and event handlers. An event handler has its private memory area cleared at the end of each release, which means that for state to persist across releases it must be saved in a memory area further up the hierarchy, usually the mission memory. But, a managed thread's memory area is only cleared when its `run()` method returns. This means that, with careful programming to avoid memory leaks, data can be stored locally and preserved over successive application-implemented 'releases' of the thread.

The thread's memory area can last for as long as the memory area of its controlling mission, which is where persistent data used by an event handler is normally stored (although it could be stored in a memory area higher up the memory hierarchy, like immortal memory). However, this ability to encapsulate state is important from a software engineering perspective, since storing a schedulable's private data in the memory area of its controlling mission makes this data more widely visible than it should be.

As an example, we consider several schedulables that log changes to the system's environment into local bounded buffers. When a buffer becomes full (which may take several releases), the data is copied into a single global buffer in mission memory, which another schedulable uses to write the system log to secondary storage.

If the logging schedulables are event handlers, then the local buffers need to be stored in mission memory because the event handler's private memory areas are cleared at the end of each release. Using managed threads, the local buffers can be stored in the private memory areas, because they are not cleared until their associated managed threads terminate. This means that the buffers do not become exposed to access by other schedulables.

```
Runnable runWork = new Runnable () {
  public void run() {
    work();
  }
};

public final void run() {
  if (waitFirstRelease()) {
    while(!myMission.terminationPending()) {
      nextRelease.add(period,100);
      ManagedMemory.enterPrivateMemory(privateMemorySize, runWork);
      nextDeadline.add(period,100);
      deadlineMissDetection.scheduleNextReleaseTime(nextDeadline);
      Services.delay(nextRelease);
    }
  }
}
```

Figure 3.8: Augmented Periodic Schedulable Object

Application-implemented releases, such as those discussed in Sect. 3.3.1, can be augmented with a nested private memory area to provide a location for objects that can be deallocated at the end of each application-implemented release. This allows a thread that can store local data until it is terminated and allocate temporary objects during a 'release'. This is illustrated in Fig. 3.8, which just shows the augmented `run()` method (and an associated runnable) of Fig. 3.5. This would provide more efficient use of memory if the thread allocates temporary objects during its application-implemented release.

3.4 SCJ Level 2 Challenges

During our investigation of Level 2's features, we have identified three challenging areas where SCJ's support could be improved. In this section we discuss these challenges and propose changes to the SCJ API to improve the support in that area. First, in Sect. 3.4.1, we discuss a problematic corner case involving thread termination. This problem was present in SCJ v0.94, but has since been fixed, after we identified it in [89]. Section 3.4.2 discusses the idea of giving deadlines to a mission sequencer's `getNextMission()` method. Finally, Sect. 3.4.3 discusses an approach to adding support for compositional timing analysis to SCJ.

3.4.1 Managed Thread Termination

In SCJ, a managed thread terminates when it returns from its `run()` method. Level 2 programs may suspend their execution, which the discussion in Sect. 3.3 shows to be useful in SCJ. We found that, in a previous version of the draft SCJ language specification [78] (v0.94), the combination of managed threads and suspension could lead to a problematic corner case during mission termination where a thread might not terminate.

The previous version of SCJ defined the following activities to be performed on receipt of a mission termination request:

- invoke this mission's `terminationHook()` method;

- invoke the `signalTermination()` of each of this mission's registered schedulables;
- disable each of this mission's periodic event handlers, so that no further firings occur;
- disable each of this mission's aperiodic event handlers, so that no further firings are honoured;
- clear the pending release event (if any) for each of this mission's event handlers so that they can be terminated after completing any active event handling;
- wait for all of this mission's schedulables to terminate;
- invoke the `cleanUp()` method for each of this mission's registered schedulables; and,
- invoke the `cleanUp()` method of this mission.

We note that the termination activities do not include invoking `interrupt()` on each of the schedulables, which would result in waking all blocked schedulables with an exception and expedite termination. This must be programmed using the `Mission.terminationHook()` method, which can be inconvenient when the mission has many schedulable objects.

As an example, we reconsider the event-released periodic thread in Sect. 3.3.1, which shows the example of a periodic thread that waits for its first release using the `Object.wait()` method. However, if the periodic thread is waiting when its controlling mission begins termination, then it may not finish its current release – its `run()` method may remain active.

To aid the termination of managed threads (and other schedulables) that are suspended when a termination request is received, we proposed (in [89]) that either the SCJ infrastructure interrupts each of the mission's registered schedulables or that all the mission's schedulables are informed of a pending termination request. The latter proposal can be achieved via a new method (`terminationSignalled()`), which each schedulable must implement. The intention of this method is to allow the programmer to manually interrupt those schedulables that may be blocked when mission termination is signalled.

In the current version of the language specification [79] (v0.100), Section 3.3.6 now contains specific guidance regarding termination in Level 2 programs. The approach taken by SCJ is that programmers should manually interrupt schedulables that may be suspended using the `signalTermination()` method, which is called on each of a mission's registered schedulables during termination. The `Mission.terminationHook()` method has been removed. This partial adoption of out proposal does not automatically solve the problem, but it does provide a uniform way of handling custom termination behaviour.

3.4.2 Deadlines on Mission Sequencers

Section 3.2.1 identifies that mission sequencers can be used to program multiple-mode applications, and that Level 2 provides more flexibility in such applications. However, an SCJ mission sequencer does not have any release parameters, so it cannot have an associated deadline or deadline-miss handler. Multiple-mode systems often have deadlines associated with their mode changes. Level 2 would be improved by the addition of these features, for situations where a mode change does not occur promptly.

```
@SCJAllowed(Level_1)
public final void requestTerminationOfCurrentMission(AbsoluteTime deadline,
     AperiodicEventHandler deadlineMiss);

@SCJAllowed(Level_1)
public final void requestMissionChange(AbsoluteTime deadline,
    AperiodicEventHandler deadlineMiss);
```

Figure 3.9: Proposed New Methods for the `MissionSequencer` Class

Adding aperiodic release parameters to mission sequencers undermines the mission programming model, particularly where they control a single non-terminating mission. Instead, we propose adding the methods shown in Fig. 3.9 to the `MissionSequencer` class. These methods invoke a mission change or termination with a deadline and a deadline-miss handler.

Both methods behave as `Mission.requestTermination()`, but provide additional behaviour. The `requestTerminationOfCurrentMission` method sets a timer, which counts down to `deadline`. If the timer expires before the mission's termination is complete, then it releases `deadlineMiss`. If the mission terminates before the timer expires, then it is cancelled.

Similarly, the `requestMissionChange` method sets a timer to count down to `deadline`. However, this method's miss-handler is released if the mission change does not occur before the timer expires. This approach allows SCJ to implement deadlines on mission changes and terminations, without altering their standard protocols. We note that these facilities might also prove useful at Level 1.

3.4.3 Support for Compositional Timing Analysis

Section 3.2.2 identifies a role for mission sequencers as a mechanism for the composition of systems from independently-developed subsystems. We represent a subsystem in SCJ with a mission sequencer controlling a single mission, which controls that subsystem's schedulables, as detailed in Sect. 3.2.2. Hence, we consider that the mission sequencer is the top component of the subsystem.

Hierarchical scheduling (and its associated schedulability analysis) is a well established technique that facilitates composition of components that have real-time attributes, such as deadlines. Unfortunately, hierarchical scheduling is supported by neither SCJ nor the RTSJ, possibly because of the lack of support by real-time operating system vendors.

We propose using two elements of hierarchical scheduling in SCJ to improve its support for independently-developed subsystems and components: CPU budgets, to implement execution-time servers; and multi-level priorities, to isolate the scheduling of subsystems. Developing an SCJ program made of subsystems can be achieved broadly in three steps.

First, each subsystem is allocated an execution-time server, which is given a capacity, a priority order, and a replenishment period. These parameters need to be assigned carefully to obtain good schedulability [25].

Second, the priority ordering of the schedulables in each subsystem is determined.

```
public class ProcessingGroupParameters {
  public ProcessingGroupParameters (HighResolutionTime start,
             RelativeTime replenishmentPeriod, RelativeTime budget){
             ...
  }
  ...
}
```

Figure 3.10: Sketch of a `ProcessingGroupParameters` class for SCJ

Third, an integration step assigns concrete priorities to the schedulables based on their priority ordering and the priority order of their server.

The result of this process is that the schedulables within a subsystem are only scheduled for execution (and in priority order) when their execution-time server server has the highest priority and has available capacity. Once the parameters of the execution-time servers and the schedulables are set, schedulability analysis must be performed for each subsystem and the system as a whole. In the rest of this section we describe the integration of our proposal into SCJ. We consider only two tiers in the program hierarchy here, for brevity.

CPU Budgets

The first aspect of hierarchical scheduling required is that each subsystem is allocated a budget of CPU execution time, which is consumed whenever one of its schedulables is executing, and a period after which its budget is replenished. When a subsystem's budget is exhausted, all of its associated schedulables are suspended until its next replenishment. In the RTSJ, this functionality can be supported by processing groups, if all the schedulables run on the same CPU.

Processing groups ensure that members of a group, collectively, are not given more CPU time per period than their group's budget. The RTSJ defines an optional class called `ProcessingGroupParameters`, an instance of which is associated with each schedulable in the processing group. This allows the RTSJ's schedulables to share a budget while retaining their individual priorities, deadlines, and periods.

Processing groups support the requirements of compositional timing analysis. So a possible solution is for SCJ to implement the class sketched in Fig. 3.10, which is a restricted version of RTSJ's `ProcessingGroupParameters`. Here, the processing group's deadline is equal to its replenishment period. However, this technique inherits the limitation that the schedulables within a subsystem must execute on the same processor.

Simulating Multi-Level Priorities

The second aspect of hierarchical scheduling that we require is multi-level priorities, which can be simulated in SCJ by manipulating the priorities of mission sequencers and schedulables. For each mission sequencer, we propose:

- defining a priority range, from the priority of this mission sequencer to the priority of the next highest priority mission sequencer, and;

- setting the priorities of all the schedulables in this subsystem within this range, while maintaining their original priority order, to ensure that they only run when their subsystem has the highest priority of all the subsystems.

This priority manipulation is performed statically, before the program is executed, in the integration step of our proposed process. It may be that priorities of the program's mission sequencers must be changed during integration to accommodate the schedulables. This is allowed, as long as the priority order of the mission sequencers is maintained.

For example, we consider a simple two-subsystem program using rate-monotonic scheduling (where higher priorities are assigned to tasks with shorter periods). The parameters of the execution-time servers of each subsystem are shown in Table 3.1. At the top level, the

	Period (ms)	Budget (ms)
Server 1	100	40
Server 2	50	15

Table 3.1: Execution-Time Server Parameters

execution-time server *Server 1* has a replenishment period of 100 milliseconds and a budget of 40 milliseconds. The execution-time server *Server 2* has a replenishment period of 50 milliseconds and a budget of 15 milliseconds. The top-level is schedulable when the priority of *Server 2* is greater than the priority of *Server 1*.

	Schedulable	Priority
Server 1		10
	S1	10
	S2	11
	S3	12
Server 2		20

Table 3.2: Execution-Time Server and Schedulable Priorities

The subsystem associated with *Server 1* contains three schedulable objects, *S1*, *S2*, and *S3*. They require a priority ordering where *S3* has a higher priority than *S2*, which has a higher priority than *S1*. During system integration, the priorities of the servers and schedulables could be assigned so that the priority of *Server 2* is greater than that of *Server 1*, plus at least 3, in order to allow the priorities of the schedulable objects to be assigned between those of the two servers. Table 3.2 shows an example of the priorities of this system, assigned to simulate multi-level priorities, because the schedulables of the subsystem associated with *Server 1* are not able to run if *Server 2* is executing.

Incorporation into SCJ

As detailed above, to support CPU budgets, SCJ needs to implement processing groups, and SCJ can already support multi-level priorities, by manipulating the priorities of an application's schedulables and mission sequencers. To aid the integration of these two aspects of hierarchical scheduling into SCJ applications, a new subclass of mission sequencer can be added to encapsulate the concerns of a subsystem, as sketched in Fig. 3.11.

```
public class Subsystem extends MissionSequencer{
    public Subsystem (PriorityParameters pri, StorageParameters storage,
                ProcessingGroupParameters params, int priRange){
            ...
  }
  ...
}
```

Figure 3.11: The `Subsystem` Class, which Provides an Interface for Subsystems

The constructor in Fig. 3.11 takes a `ProcessingGroupParameters` object (Sect. 3.4.3). To encapsulate the information needed for the priority manipulation described in Sect 3.4.3, the values of `pri` (which is the priority of this mission sequencer) and of `pri + priRange` define the priority range for schedulable objects encapsulated by this subsystem.

3.5 Summary

This chapter presents the first investigation of the utility of Level 2's features. The three unique features that we examine – nested mission sequencers, managed threads, and suspension – all have uses in SCJ. Nested mission sequencers are useful for programming multiple-mode applications and independently-developed subsystems; Levels 0 and 1 cannot. Managed threads can be used to program background tasks and provide better encapsulation of state than SCJ's event handlers. Suspension enables Level 2 to capture systems that require it, such as producer-consumer systems. When combined, managed threads and suspension can provide extended release patterns.

Level 2's unique features, and some of the use cases we identify, have revealed some challenges for Level 2 programs. In v0.94 of the SCJ Language Specification [78] there was a problematic corner case where suspended schedulables were not interrupted during the termination of their controlling mission. We proposed that all schedulables are interrupted during mission termination, or that a method call be used to signal a mission's registered schedulables. The current version of the language specification [79] (v0.100) guides programmers to manually interrupt schedulables that may be suspended using the `signalTermination()` method, which is called on each of a mission's registered schedulables during termination.

The second challenge was the inability to place a deadline on a mission sequencer changing or terminating missions. We propose adding two methods to the `MissionSequencer` class to request the termination or change of a mission, giving a deadline and a deadline-miss handler for the process. This is yet to be added to SCJ.

The final challenge is the lack of support for compositional timing analysis of SCJ programs. To address this difficulty, we propose three things. First, the implementation of a version of the RTSJ's processing group parameters to support CPU budgets. Second, a technique for manipulating the priorities of a program's schedulables to simulate multi-level priorities. Third, to aid integration we propose a subclass of mission sequencer to encapsulate a subsystem. However, these proposals are also yet to be integrated into SCJ. In the next chapter, we describe a model of SCJ Level 2. We do not include the unadopted proposals: our model is faithful to v0.100 of the SCJ Language Specification.

Chapter 4

Safety-Critical Java Level 2 Modelling Approach

This chapter describes our approach to modelling the paradigm of Safety-Critical Java (SCJ) Level 2 v0.100 [79]. Our model is written in the state-rich process algebra *Circus*; a primer for which is provided in Sect. 2.4. Section 4.1 introduces at a high level our approach to modelling the SCJ Level 2 paradigm. Section 4.2 details the components of our model, showing how we capture Level 2's features. As described in Sect. 1.2, our model abstracts away from scheduling and resources, which means that it does not capture SCJ's global multi-processor support, task scheduling, or region-based memory management. The full model of the SCJ Level 2 paradigm can be found in Appendix C.

In Sect. 4.3 we describe how our model captures synchronisation and suspension, and Sect. 4.4 describes how our model captures inheritance and polymorphism. Section 4.5 discusses using our model to show that the termination protocol from SCJ v0.94 was overly complicated, and the current termination protocol (which we proposed in [56]) is simpler. Finally, Sect. 4.6 summarises our modelling approach.

4.1 Modelling Overview

We provide the first formal semantics of the SCJ Level 2 programming paradigm, as described in the SCJ language specification. We capture the paradigm of SCJ v0.100 [79], as described in Chap. 2. We take the view that this paradigm is separate to its realisation in Java; we capture the paradigm, abstracting away from the implementation details.

Our model is beneficial for both top-down and bottom-up SCJ development. Top-down, our model provides a target for the *Circus* refinement strategy [17] to cater to Level 2 programs. The strategy allows refinement of abstract specifications of behaviour into concrete specifications that capture the SCJ paradigm. Combining our model with the strategy would allow the development of SCJ Level 2 programs that are correct-by-construction. Bottom-up, our model can be used to aid program verification. Level 2 programs can be translated into our model (which we discuss further in Chap. 5) for analysis. We can then use the model of a program to catch program errors, such as deadlock, livelock, and exceptions.

Our overall modelling approach is illustrated in Fig 4.1. Firstly, we capture the behaviour of the SCJ API (as described in the SCJ language specification) in the component called

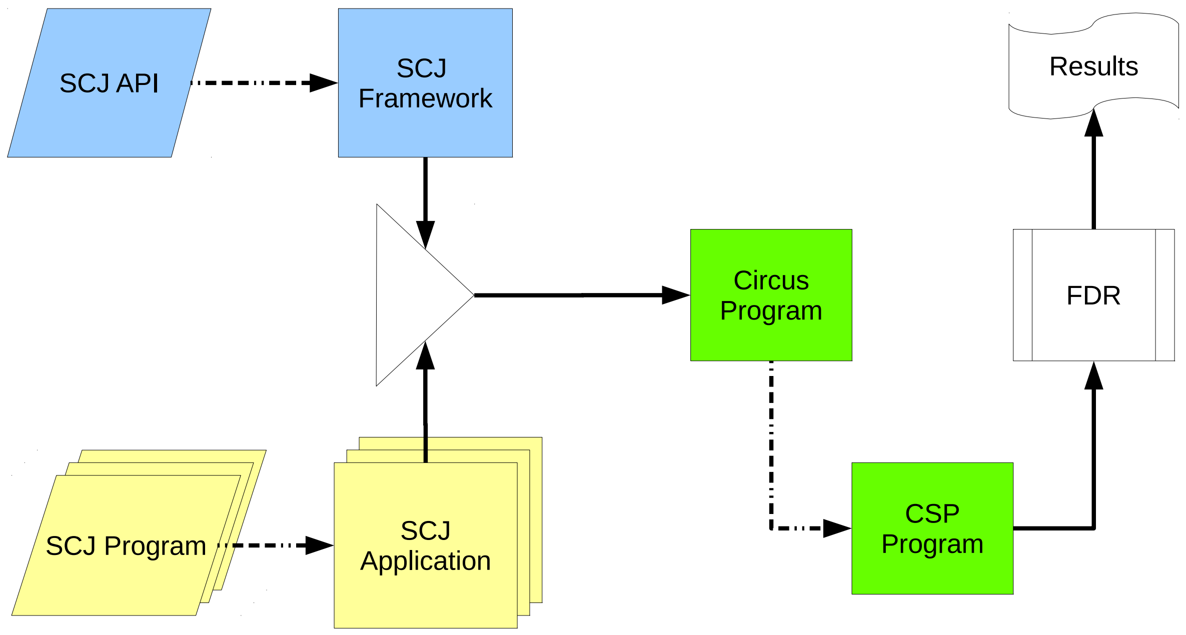


Figure 4.1: High-Level Modelling Approach

the framework model. The framework model is generic, in that it captures the behaviour common to all programs, and is reused when modelling each new program. Next, we capture the program-specific behaviour of a particular program in a component called the application model, which must be generated afresh for each input program.

These two models are combined to form a *Circus* program that captures the behaviour of an input program. The separation of the generic from the program-specific behaviour in our model has two main benefits. First, the framework model can be validated in isolation, before we start checking programs for errors. Second, it simplifies the application model and, therefore, the translation from an SCJ program to its application model. The communication and cooperation between these two models is discussed in more detail in Sect. 4.2.

To verify programs using this technique, we must first translate our model from *Circus* to CSP_M – machine readable CSP, which is the input language for FDR3. Then we can use FDR3 to check the model for deadlock, livelock, and divergence. The latter check also shows if the model may throw an exception, because of the way we model exceptions. The extra translation step is needed because there is currently no model checker for *Circus*. However, it has the added benefit of decoupling the *Circus* model from the CSP_M version, which is adapted to enable tractable analysis in FDR3.

Our approach is based on that of a model of SCJ Level 1 [93], but instead of simply adding Level 2 features to this model we also capture Level 1 features that it did not: exceptions, synchronisation, and period or deadline overrun. We also, in contrast to the Level 1 model, provide separate framework processes for each of the three event handlers, which considerably simplifies their application processes and translation. Further, our model raises an exception if a schedulable is registered twice, as specified by the API. Finally, since our model is based on a newer version of the SCJ Language Specification, we capture API updates that were implemented after the model of Level 1 was constructed.

Our model of Level 2 is commensurately larger than the Level 1 model [93]. Table 4.1 summarises the relative sizes of both models, showing the number of lines of *Circus* that

Component	Level 1	Level 2
Safelet	27	107
Mission Sequencer	36	153 + 236
Mission	120	360
Event Handler	49	N/A
Aperiodic Event Handler	N/A	252
Oneshot Event Handler	N/A	316
Periodic Event Handler	N/A	267
Managed Thread	N/A	360
Java Thread	N/A	145
Java Object	N/A	731
Total Lines in SCJ API	232	2671
Other Definitions	469	695
Total Lines of *Circus*	701	3366

Table 4.1: Summary of the Sizes of Level 1 and Level 2 Models

model each component and capture the other definitions. The Level 1 model comprises $\sim$ 700 lines of *Circus* and the SCJ API is captured by 4 processes, which total $\sim$ 232 lines. In contrast, our model of Level 2 comprises $\sim$ 3360 lines of *Circus* and the SCJ API is captured by 10 processes, which total $\sim$ 2600 lines. The Level 1 model uses one process for all three event handler classes, whereas our Level 2 model uses three separate processes to capture these three classes. Also, mission sequencers are modelled by two separate processes in our Level 2 model, as opposed to a single process in the Level 1 model.

Each class in the SCJ library and object in the program is represented in our models by a *Circus* process. We use *OhCircus* classes to capture non-reactive behaviour, such as methods that are purely data operations. *OhCircus* classes are similar to Java classes: they may hold variables, specify constructors, make use of inheritance, and must be instantiated before use. Specifically, data operations are captured in methods, which may be called from processes. We also use constructs from *CircusTime* to capture deadlines and timed offsets.

Each process takes the name of the class it models, suffixed with 'FW' for framework or 'App' for application processes. Each application process is parametrised by a unique identifier, which is used to identify the process in channel communications and allows framework processes to communicate with their application counterparts. The exception to this is the $SafeletFW$ process, which only has one instance because there is only one safelet in an SCJ program. Multiple instances of the same class in a program each have their own identifier.

Each of an object's methods are modelled by an action in the process that represents that object. Calls to and returns from a method are represented by a pair of channels, which signal the start and end of the action modelling the method. These channels take the name of the method, suffixed with '$Call$' for a channel representing the call to the method, or 'Ret' for a channel representing the return from the method. In the framework model, some of the API methods are simple enough that they are modelled by only one channel. For example, the `Mission.missionActive()` method, which returns a boolean representing if this mission has been started or not. Using channels to represent calls to and return from methods allows method calls between processes. We handle method calls between processes using a separate process to 'route' the calls to the right process, which we discuss in Sect. 4.4.

We capture exceptions, but not the Java exception handling mechanism. Exceptions are used in two ways in the SCJ language specification: as an artefact of SCJ's implementation in Java and as a way of indicating misuse of the paradigm. We model the latter category, as we consider that exceptions are used in lieu of preventing paradigm misuse. We model these exceptions with the *throw* event followed by **Chaos**, which is an in-built divergent process that allows us to detect exceptions during analysis.

We capture exceptions thrown by the API when: a thread is interrupted, a method receives an inappropriate argument, a thread attempts to use suspension without holding the lock, a thread attempts to lock an object with a priority lower than the thread's, or a mission attempts to register a schedulable that is already registered to another or the same mission. We also capture exceptions thrown by the Java `assert` statement, to support programs that use it to thrown application-specific exceptions that indicate a misuse of the paradigm.

4.2 Model Structure

As previously mentioned, we model the state and behaviour of application objects in the program as two cooperating components: the framework model and the application model. The processes in these two components must communicate with each other to form the full model of a program. This section discuses the communication and cooperation between the framework and application processes representing the main SCJ API classes.

Figure 4.2 shows the main processes in the framework model and the channels that they use to communicate. The channels with underscores in their names are control signals (for example, *start_mission*) and those in camel case represent method calls (for example, *initializeCall* and *initializeRet*). Some of the channels have been omitted for brevity, indicated by three dots. The layering indicates potentially multiple instances in one model. Each of these framework processes communicate with an application process, which are also not shown in Fig. 4.2.

The framework and application models cooperate to represent the behaviour of the input program. When a framework process encounters application-specific methods, it signals its application counterpart to take control and perform that method's behaviour. Another signal returns control to the framework. These signals are call-return event pairs that retain the method name, suffixed with '*Call*', for the event modelling the method call, or '*Ret*', for the event modelling its return.

We illustrate this in Fig. 4.3, which shows how we model a mission `MyMission` using an instance of the *MissionFW* process and an instance of *MyMissionApp*. The `MyMission` class implements the `initialize()` and `cleanUp()` methods, which it inherits from the `Mission` class. The instance of *MissionFW* contains actions representing these two methods; since their default implementations contain no behaviour, the actions representing those methods simply contain the associated pair of call and return events.

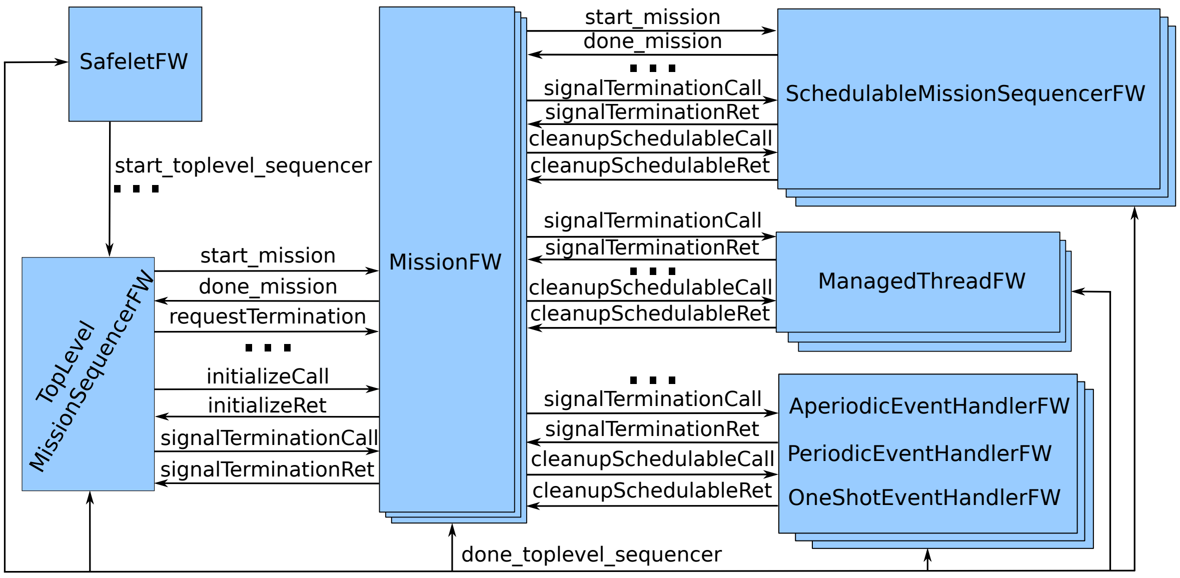


Figure 4.2: Level 2 Model Structure

The instance of *MyMissionApp* also contains actions representing the `initialize()` and `cleanUp()` methods. However, because *MyMissionApp* models the behaviour of `MyMission` the actions it contains describe the full behaviour of the methods from `MyMission`. Because the two processes are associated via a shared identifier, they can communicate so that the *MissionFW* process can hand control to the *MyMissionApp* process to achieve a model of the behaviour of `MyMission`.

In the rest of this section we discus our models of the API classes and how they communicate with their application counterparts to model a program. We use the aircraft control system, described in Sect. 2.1.3, to illustrate how we model an application. Section 4.2.1 provides a more detailed examination of the modelling pattern, because the safelet is relatively simple. But the description in the other sections below needfully omits some of the behaviour that is not pertinent to the main control flow of SCJ programs.

4.2.1 Safelet

The framework process *SafeletFW*, shown in Appendix C.8, handles the behaviour specified by the `Safelet` interface. It is the process that defines the main execution flow of the program. It contains an action modelling the generic behaviour of the `getSequencer()` method, which gets the identifier of the top-level mission sequencer from its counterpart application process and signals that it should start.

Additionally, the *SafeletFW* process tracks which schedulables in the program are registered to a mission. This collection of globally registered schedulables is used to trigger a *MissionFW* process to raise an exception if it attempts to register a schedulable that is already registered. This behaviour can be seen in the communication between the the *SafeletFW*'s *Register* action (in Appendix C.8) and the *MissionFW*'s *Register* action (in Appendix C.10) over the *checkSchedulable* channel. The SCJ Language Specification states that this exception should be raised, but it is not part of the safelet's specification. The safelet was chosen to keep track of which schedulables are registered globally because it is at the top of the program hierarchy.

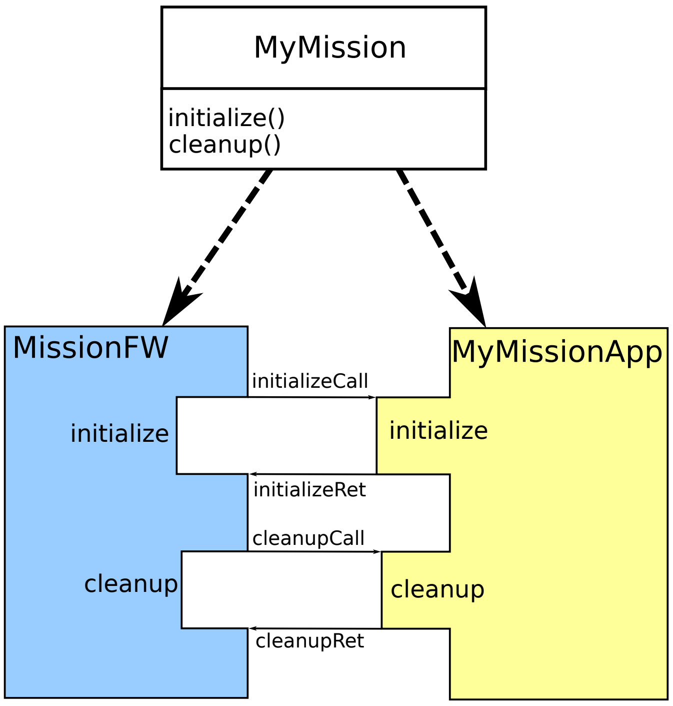


Figure 4.3: Modelling a Mission Using Two Components

The application model of a class that implements the `Safelet` interface provides the application-defined behaviour of the `getSequencer()` method. This method returns a reference to the top-level mission sequencer object. We model it using an action that returns the identifier of a mission sequencer process. The safelet application model also contains actions modelling any application-defined methods.

4.2.2 Mission Sequencers

Two framework processes model the behaviour of the `MissionSequencer` class, because it may be used in two different contexts. The *TopLevelMissionSequencerFW* process (shown in Appendix C.9) models the top-level mission sequencer and the *SchedulableMissionSequencerFW* process (shown in Appendix C.11) models a nested mission sequencer, which is used as a schedulable – as described in Sect. 3.2. This simplifies both processes because they each only need to be involved in events relevant to their context.

The *TopLevelMissionSequencerFW* process is started by the *SafeletFW* process. When it begins termination, it signals the *SafeletFW* to indicate that the program has terminated. A *SchedulableMissionSequencerFW* process is started by its controlling mission, and signals to its controlling mission once it has terminated. Since it is a schedulable, it must respond to termination requests from either its controlling mission or the mission it is executing.

The framework models of both flavours of mission sequencer contain an action modelling the generic behaviour of the `getNextMission()` method. This method returns a reference to the next mission that this mission sequencer executes. The action modelling the method fetches a mission identifier, from the application counterpart of this framework process, and starts that mission.

Despite having two separate framework models, the application model of either a top-level or a nested mission sequencer fits the same pattern. The application model of a mission

$$\begin{array}{l} \mathit{Register} \mathrel{\widehat=} \\ \quad \mathit{register} \mathbin{?} s \mathbin{!} \mathit{mission} \longrightarrow \\ \quad \left(\begin{array}{l} \left(\begin{array}{l} \mathit{checkSchedulable} \,.\, \mathit{mission} \mathbin{?} \mathit{check} : (\mathit{check} = \mathbf{True}) \longrightarrow \\ \mathit{AddSchedulable} \end{array}\right) \\ \Box \\ \left(\begin{array}{l} \mathit{checkSchedulable} \,.\, \mathit{mission} \mathbin{?} \mathit{check} : (\mathit{check} = \mathbf{False}) \longrightarrow \\ \mathit{throw}.\mathit{illegalStateException} \longrightarrow \\ \mathbf{Chaos} \end{array}\right) \end{array}\right) \end{array}$$

Figure 4.4: The *MissionFW*'s *Register* Action

sequencer provides the application-defined behaviour of the `getNextMission()` method: the behaviour that returns a mission identifier. The application model of a mission sequencer also contains actions representing any application-defined methods.

4.2.3 Mission

A *MissionFW* process is started by a mission sequencer process and models the generic behaviour of the `Mission` class. It has actions that model the methods `initialize()`, `requestTermination()`, and `cleanUp()` from the `Mission`. Here we present the *Circus* model of a mission in more detail than the processes of the other paradigm objects, because it is ideal for illustrating our modelling approach.

The *InitializePhase* action models the `initialize()` method, and controls the schedulables being registered to this mission. The events *initializeCall* and *initializeRet* model the call to and return from `initialize()`. *InitializePhase* comprises other actions that each control a particular part of the registration protocol. For example, the *Register* action (shown in Fig. 4.4), which controls the registration of one schedulable.

The *Register* action (Fig. 4.4) is triggered by the *initializeCall* event, and is then ready to register schedulables. The registration of a schedulable is modelled by the event $\mathit{register} \,.\, s \,.\, m$, where s is the identifier of the schedulable being registered and m is the identifier of the mission registering the schedulable. The *Register* action accepts a *register* event, with any schedulable identifier as long as the mission identifier matches the identifier of this *MissionFW* process. These *register* events originate in the *MissionFW*'s application counterpart.

Next, *Register* waits for the *checkSchedulable* event, which indicates, via the variable *check*, if a schedulable may be registered. This event comes from the *SafeletFW* process, which listens to all *register* events and tracks globally registered schedulables. This allows the detection of an attempt to register a schedulable more than once. If *check* is **True**, then the schedulable can be registered. If *check* is **False**, then the schedulable is already registered and we use the *throw* channel to model an exception being thrown and then diverge (**Chaos**).

Once all of a mission's schedulables have been registered, the *MissionFW* process enters its execution phase, where it starts all of its registered schedulables simultaneously and then waits to handle method calls and schedulable termination. This is captured by the *Execute*

$$\begin{array}{l}
\mathit{InitializePhase} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mathit{initializeCall} \,.\, \mathit{MainMission} \longrightarrow \\
\mathit{register} \,!\, \mathit{ProducerSID} \,!\, \mathit{MainMission} \longrightarrow \\
\mathit{register} \,!\, \mathit{ConsumerSID} \,!\, \mathit{MainMission} \longrightarrow \\
\mathit{initializeRet} \,.\, \mathit{MainMission} \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}$$

Figure 4.5: The *MainMissionApp*'s *InitializePhase* Action

action, shown in Appendix C.10. The signal to terminate a mission is accepted from any of that mission's registered schedulables, and triggers the termination and clean up of that mission's active schedules.

Once the termination and cleanup of a mission's schedulables is complete, it enters its own cleanup phase. This is controlled by the *Cleanup* action, which models the `CleanUp()` method. This simply indicates to the *SafeletFW* process that this mission's schedulables are no longer registered and triggers the *Cleanup* action of this mission's application counterpart.

As an example of a mission application process, we examine the model of the `MainMission` class in the Buffer application, described in Sect. 2.1.2. The `MainMission` is modelled by the *MainMissionApp* process, which captures the reactive application-specific behaviour of the mission. It cooperates with an instance of the *MissionFW* process to capture the behaviour of the mission. Channels on which the instances of *MissionFW* and *MainMissionApp* communicate are parametrised by the mission identifier *MainMission*; this ensures that the *MainMissionApp* process communicates with the right process instance. These channels are used by the *MissionFW* process to hand control to the *MainMissionApp* process so that it can execute application-specific behaviour.

The application model of a mission provides the application-defined behaviour of the methods `initialize()` and `cleanUp()`. The `initialize()` method is modelled by the *InitializePhase* action, and clearly illustrates the simplification of the application model of a mission in comparison to its framework model.

In an SCJ program, the `Mission`'s `initialize()` method is overridden to register the schedulables that this particular mission controls. Figure. 4.5 shows the *InitializePhase* action of the *MainMissionApp* process, which models the `initialize()` method in the `MainMission` class (shown in Appendix A).

The *InitializePhase* action is triggered by the *initializeCall* event and then proceeds to register schedulables, which is modelled by the same event as in the *MissionFW* process, $\mathit{register} \,.\, s \,.\, m$. The *register* events shown in Fig. 4.5 send the identifiers *ProducerSID* and *ConsumerSID*, respectively, to the *MissionFW* process with the identifier *MainMission*. The order of registration shown in Fig. 4.5 corresponds to the order in the program.

The `cleanUp()` method is modelled by the *CleanupPhase* action. In the buffer application, there is no application-defined behaviour in the `cleanUp()` method, so the *CleanupPhase* action simply performs the call and return events. In a program with cleanup behaviour, events modelling it would be inserted in-between the call and return events.

$$PriorityQueue == PriorityLevel \to (\operatorname{iseq} ThreadID)$$

$$\forall pq : PriorityQueue \bullet nullThreadId \notin \operatorname{ran}(\bigcup(\operatorname{ran} pq))$$

Figure 4.6: Priority Queue Function

4.2.4 Schedulables

The generic behaviour of each of the five schedulables is modelled by a different process: the *PeriodicEventHandlerFW* process (shown in Appendix C.12) models a periodic event handler, the *AperiodicEventHandlerFW* process (Appendix C.13) models either an aperiodic event handler or an aperiodic long event handler, the *OneShotEventHandlerFW* process (Appendix C.14) models a one-shot event handler, the *ManagedThreadFW* process (Appendix C.15) models a managed thread, and the *SchedulableMissionSequencerFW* process (Appendix C.11) models a schedulable mission sequencer. Each schedulable process is started by a mission, performs its behaviour, accepts termination requests from the mission that started it, and cleans up after it terminates.

Each event handler process has actions that control its specific release pattern. The release behaviour of an event handler class is contained in its `handleAsyncEvent()` method, which is modelled by an action that is triggered in accordance with the handler's release pattern.

Event handlers may have deadlines, and periodic event handlers have a period. Our models consider that periods may be overrun and deadlines may be missed, and captures the response to this. This means that our model is capable of being used for checking potential deadline or period overruns, for example. However, we note that implementing these check is left as future work – as discussed in Sect. 6.3.

Managed threads are simpler and begin their release as soon as they are started. The release behaviour of a managed thread is contained in its `run()` method, the generic behaviour of which is modelled by an action that is called as soon as the managed thread is started. Mission sequencers used as schedulables are described in Sect. 4.2.2.

Each of the schedulable application processes has an action modelling the method that captures its release behaviour. For an event handler, this is the `handleAsyncEvent()` method. For a managed thread, this is the `run()` method. Because the release patterns of the event handlers are captured by their framework processes, the application models for these classes are very simple; they only require the release behaviour. The application models of the schedulables also contain actions modelling any application-defined methods.

4.3 Synchronisation and Suspension

The synchronisation model of SCJ constrains that of standard Java. First, SCJ programs cannot use `synchronized` blocks, only `synchronized` methods. Second, threads queue for a lock in order of eligibility. In SCJ, the most eligible thread is the thread at the highest priority level that has been waiting for the longest time. We model this using the type *PriorityQueue* (Fig. 4.6), which is a total function from *PriorityLevel* to injective sequences of *ThreadID*.

PriorityLevel is a free type containing the priorities available to the system and *ThreadID* is the set of thread identifiers.

Our models use extra framework processes to control synchronisation and suspension. In SCJ, each schedulable is executed by a thread. In our model, schedulables that call a synchronised method are associated with an instance of the *ThreadFW* process – shown in Appendix C.6. *ThreadFW* holds the thread identifier and keeps track of its priority and interrupted status. Overall, the framework model of a schedulable that calls a synchronised method is the parallel composition of its associated *ThreadFW* process with the appropriate framework process, which depends on the type of schedulable (event handler, managed thread, or mission sequencer).

Additionally, each object used as a lock is associated with an instance of the *ObjectFW* process, which stores the threads waiting on this object and controls the threads trying to lock this object. Again, the overall framework model of each object that represents a paradigm component and is used as a lock is its framework process in parallel with an instance of *ObjectFW*. Non-paradigm objects used as locks are modelled in the framework by just an instance of *ObjectFW*.

As an example, we revisit the buffer application described in Sect. 2.1.2, in which a producer and consumer thread each share access to a bounded buffer. The `Buffer` object controls access to the buffer with the methods `read()` and `write()`, which are synchronised, to control the concurrent access. We consider the `read()` method here as an example of our approach to modelling synchronisation and suspension. The `read()` method, shown in Fig. 4.7, suspends the calling thread (by calling `wait()`) if the buffer is empty. This is wrapped in a loop that checks if the buffer is empty, to deal with spurious wake ups by the Java Virtual Machine.

The instance of *ObjectFW* that is associated with the *BufferApp* process (which models the reactive behaviour of the `Buffer` object) controls the synchronisation and suspension behaviour using the *startSyncMeth*, *lockAcquired*, and *endSyncMeth* events. The *startSyncMeth* event models the beginning of a synchronized method and triggers the *ObjectFW* process to request a lock on this object by the thread calling this action.

Because the lock may already be held by another thread, the *readSyncMeth* action waits for the *lockAcquired* event (from the *ObjectFW* process) to signal that it has the lock and can proceed. After the body of the method, the *endSync* event signals that the synchronised method is complete, to trigger *ObjectFW* to release the lock on the mission currently held by the calling thread. We note that SCJ does not support Java's `ReentrantLock`, however, SCJ does support reentrant locking by allowing synchronised methods to call other synchronised methods in the same object. The *ObjectFW* process provides this behaviour; to unlock the object, after the first *lockAquired* event, each subsequent *startSyncMeth* event (which must be from the same thread) must be matched by an *endSyncMeth* event from the locking thread.

We model the call to `wait()` using the call-return event pair *waitCall* and *waitRet*. These events take the identifier of the associated *ObjectFW* instance (*BufferOID*, in Fig. 4.7) and the identifier of the *thread* calling this action. The instance of *ObjectFW* associated with the mission adds *thread* to its queue of waiting threads. The process calling *waitCall* waits for *waitRet* to communicate its identifier.

$$
\begin{array}{l}
\mathit{readSyncMeth} \mathrel{\widehat=} \mathbf{var}\, \mathit{ret} : \mathbb{Z} \bullet \\
\left(\begin{array}{l}
\mathit{readCall} \,.\, \mathit{BufferID} \,?\, \mathit{caller} \,?\, \mathit{thread} \longrightarrow \\
\left(\begin{array}{l}
\mathit{startSyncMeth} \,.\, \mathit{BufferOID} \,.\, \mathit{thread} \longrightarrow \\
\mathit{lockAcquired} \,.\, \mathit{BufferOID} \,.\, \mathit{thread} \longrightarrow \\
\left(\begin{array}{l}
\mathbf{var}\, \mathit{bufferEmpty} : \mathbb{B} \bullet \mathit{bufferEmpty} := \mathit{bufferEmpty}(); \\
\mu X \bullet \\
\left(\begin{array}{l}
\mathbf{var}\, \mathit{loopVar} : \mathbb{B} \bullet \mathit{loopVar} := \mathit{bufferEmpty}; \\
\mathbf{if}\ (\mathit{loopVar} = \mathbf{True}) \longrightarrow \\
\quad ;\, X \\
\;[\!]\ (\mathit{loopVar} = \mathbf{False}) \longrightarrow \mathbf{Skip} \\
\mathbf{fi}
\end{array}\right) ; \\
\left(\begin{array}{l}
\mathit{waitCall} \,.\, \mathit{BufferOID} \,.\, \mathit{thread} \longrightarrow \\
\mathit{waitRet} \,.\, \mathit{BufferOID} \,.\, \mathit{thread} \longrightarrow \\
\mathbf{Skip}; \\
\mathit{bufferEmpty} := \mathit{bufferEmpty}()
\end{array}\right) ; \\
\mathbf{var}\, \mathit{out} : \mathbb{Z} \bullet \mathit{out} := \mathit{buffer}\, ; \\
\mathit{ret} := \mathit{out}
\end{array}\right) ; \\
\mathit{endSyncMeth} \,.\, \mathit{BufferOID} \,.\, \mathit{thread} \longrightarrow \\
\mathit{readRet} \,.\, \mathit{BufferID} \,.\, \mathit{caller} \,.\, \mathit{thread} \,!\, \mathit{ret} \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}\right)
\end{array}
$$

Figure 4.7: The *BufferApp* Proceess's *readSyncMeth* Action

We model the call to `notify()` with the event *notify*. As with *waitCall* and *waitRet*, *notify* also takes the identifier of the associated *ObjectFW* process and the identifier of the *thread* calling this action. The *notify* event triggers the *ObjectFW* process to resume the most eligible thread. If there are no waiting threads, then *ObjectFW* allows the call to *notify*, but does nothing. To resume a thread, *ObjectFW* calls *waitRet* with the identifier of the thread to be resumed.

SCJ Level 2 can also use the `notifyAll()` method, which resumes all the waiting threads in eligibility order. We model a call to `notifyAll()` with the event *notifyAll*. It triggers the *NotifyAll* action in the *ObjectFW* process to call the *NotifyAllHandler* action, which uses *waitRet* to resume the most eligible thread and then recurses if there are more threads waiting.

4.4 Inheritance and Polymorphism

Inheritance and polymorphism are key features of object-oriented programming. An SCJ program's paradigm objects (the safelet, mission sequencer, mission, etc) each extend an SCJ API class, except for the safelet, which implements an interface. As in standard Java, SCJ classes may extend one super class and implement one or more interfaces.

When modelling inheritance, we must consider that a method's declaration and implemen-

$$
\begin{array}{l}
read\_MethodBinder \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
binder\_readCall \,?\, l : (l \in readLocs) \,?\, c : (cs \in readCallers) \,?\, callingThread \longrightarrow \\
readCall \,.\, l \,.\, c \,.\, callingThread \longrightarrow \\
readRet \,.\, l \,.\, c \,.\, callingThread \,?\, ret \longrightarrow \\
binder\_readRet \,.\, l \,.\, c \,.\, callingThread \,!\, ret \longrightarrow \\
read\_MethodBinder
\end{array}\right)
\end{array}
$$

Figure 4.8: The *read_MethodBinder* Action

tation may be in different classes. Further, implementations may be overridden in subclasses. From what we have already described, this causes problems for our modelling approach. We describe our solution to this problem here.

As described in Sect. 4.2, our model usually captures method calls using a pair of channels representing the call to and return from the method. This pair of channels is parametrised by: the identifiers of the process representing the calling class and the process representing the called class, any parameters the method may take, and its return value – if it has one. However, the location of the method may not be the class on which the call is being made, so the action representing it may not exist in the process representing the called class, causing a spurious deadlock.

In our model of SCJ programs, we handle calls to inherited methods using a process named the *MethodCallBinder* (*MCB*), which is placed in parallel with the Application model to bind method calls to the method locations. The *MCB* also facilitates non-inherited method calls made between application processes. The *MCB* process is constructed during the translation for a particular SCJ program, as described in Chap. 5.

Since SCJ allows dynamic class loading, the *MCB* can resolve dynamic binding only once the entire program is known. Dynamically loaded classes are handled in the same way as any other potentially instantiated class, for example a mission choosing to instantiate only one of two different schedules in its initialisation phase. The potential for that class to be instantiated means that it will be included in the model.

The *MCB* process contains one binding action for each application-defined method in the program. Each binding action has two associated sets, a *Locs* set and a *Callers* set. To ensure uniqueness, the names of these sets are prepended by the name of the action they method they are binding. The *Locs* set contains the identifiers of all the processes that contain the method being bound, and the *Callers* set contains the identifiers of the processes that may call the method being bound.

Each action is triggered by an event that has the same name as the event representing the call to the method, but prepended with '*binder*'. This event accepts communications with any identifier form the *Locs* set and any identifier from the *Callers* set, these parameters ensure that the other evens in the action are related to the same method call. When triggered, the action engages in the call and return events of the method being bound. Finally the action signals the return from the bound method with an event that has the same name as the return channel, again prepended with '*binder*'.

$$
\begin{array}{l}
\mathit{deployLandingGear\_MethodBinder} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mathit{binder\_deployLandingGearCall}\,?\,l : (l \in \mathit{deployLandingGearLocs}) \\
\qquad ?\,c : (c \in \mathit{deployLandingGearCallers}) \\
\qquad ?\,\mathit{callingThread} \longrightarrow \\
\mathit{deployLandingGearCall}\,.\,l\,.\,c\,.\,\mathit{callingThread} \longrightarrow \\
\mathit{deployLandingGearRet}\,.\,l\,.\,c\,.\,\mathit{callingThread} \longrightarrow \\
\mathit{binder\_deployLandingGearRet}\,.\,l\,.\,c\,.\,\mathit{callingThread} \longrightarrow \\
\mathit{deployLandingGear\_MethodBinder}
\end{array}\right)
\end{array}
$$

Figure 4.9: *deployLandingGear_MethodBinder* Action from the Model of the Aircraft Application

As an example of the *MCB* facilitating non-inherited method calls, we revisit the buffer application described in Sect. 2.1.2, in which a producer and consumer thread each share access to a bounded buffer. The `Buffer` object controls access to the buffer with the methods `read()` and `write()`. In our model, both of these methods are represented by actions in the *Buffer* process. Because the `read()` method is called by the `Producer` thread and the `write()` method is called by the `Consumer` thread, they also each have an action in the *MCB*. Here we examine the `read()` method, which is controlled by the *readCall* and *readRet* events.

The *MCB* action that binds calls to the *readCall* and *readRet* events is shown in Fig. 4.8. The action is triggered by the *binder_readCall* event, which takes the same parameters as the *readCall* event. The process passes these parameters to the *readCall* event, which hands control over to the *read* action in the *Buffer* process. The *readRet* event signals that the *read* action is finished and is handing back control. The return parameter from *readRet* is passed to *binder_readRet*, which signals the end of the method call and returns control to the location of the call. The *readLocs* set contains the identifier of the *Buffer* process and the *readCallers* set contains the identifier of the *Consumer* process. This example shows the *MCB* providing a simple one-to-one binding; the method it is binding is only defined in one location and is only called by one object.

As an example of the *MCB* handling an inherited method call, we use the aircraft application described in Sect. 2.1.3, which is a simplified aircraft control system. The aircraft application operates in three modes: take off, cruise, and land. Each mode is represented by a mission that controls several mode-specific schedulables. Since both the take off and land modes activate the aircraft's landing gear, both modes contain a schedulable that controls the landing gear by calling methods in the mission representing that mode. Each of the missions implements the `LandingGearUser` interface, which defines the methods to operate the landing gear. Here we examine the `deployLandingGear()` method, which is controlled by the *deployLandingGearCall* and *deployLandingGearRet* events.

The *MCB* action binding calls to the *deployLandingGearCall* and *deployLandingGearRet* events is shown in Fig. 4.9. This action is triggered by the *binder_deployLandingGearCall* event, and its completion is signalled by the *binder_deployLandingGearRet* event, in the same way as described above. The difference with this example is in the *Locs* and *Callers* sets.

```
$GetNextMission \mathrel{\widehat=}$
  $getNextMissionCall\,.\,sequencer \longrightarrow$
  $getNextMissionRet\,.\,sequencer\,?\,next \longrightarrow$
  $currentMission := next;$
  $StartMission;$
  $\mathbf{if}\ terminating\ =\ FALSE \longrightarrow$
    $GetNextMission$
  $[\!]\ terminating\ =\ TRUE \longrightarrow$
    $\mathbf{Skip}$
```

Figure 4.10: The Original *TopLevelMissionSequencerFW*'s *GetNextMission* Action

The *deployLandingGearLocs* set contains the identifiers of both the *TakeOffMission* and *LandMission* processes, because the bound method is defined in both of these locations. The *deployLandingGearCallers* set contains the identifiers of each of the schedulables that control the landing gear. This allows events that represent either of the schedulables calling the `deployLandingGear()` method defned in either of the two missions and issue the return event to the caller. In this example, the two calls cannot occur at the same time (because the two missions where the methods are defined are mutually exclusive) but it illustrates how the *MCB* can handle simultaneous calls.

4.5 Simplifying the SCJ Termination Protocol

The current SCJ termination protocol restricts communication to components separated by one layer in the program hierarchy. For example, a schedulable can only signal its controlling mission to terminate, it cannot request the top-level mission sequencer to terminate. This is the result of our proposal, presented in [56], for simplifying the original termination protocol.

The original SCJ termination protocol (from SCJ v0.94 [78] and earlier) included the method `MissionSequencer.requestSequenceTermination()`, which allows a schedulable to request the termination of a mission sequencer. During the termination of a mission sequencer, it will terminate its current mission and any schedulables that mission is executing. The concern with this facility was that it could be misused by a schedulable to terminate an arbitrary mission sequencer. This complicates the semantics of the termination protocol needed to support mission termination and breaks the encapsulation of missions.

This section presents our models of the original termination protocols and describes the changes made to produce the current protocol. In Sect. 4.5.1 we describe a formal model of the original termination protocol, as presented in the SCJ language specification v0.94 [78]. Section 4.5.2 presents the model of the current termination protocol. Finally, Sect. 4.5.3 provides a comparison of these two models and summarises the changes that produced the current protocol.

We test the two protocols by model checking them and comparing the number of states in each model, to show that the original termination protocol was more complicated than

$$
\begin{array}{l}
\mathit{StartMission} \mathrel{\widehat=} \\
\quad \mathbf{if}\ \mathit{currentMission} \neq \mathit{nullMissionId} \longrightarrow \\
\qquad \left(\begin{array}{l}
\mathit{start\_mission}\,.\,\mathit{currentMission} \longrightarrow \\
\mathit{initializeRet}\,.\,\mathit{currentMission} \longrightarrow \\
\left(\begin{array}{l}
\mathit{RequestSequenceTermination} \\
\quad [\![\{\mathit{terminating}\} \mid \{\!|\ \mathit{end\_termination}\ |\!\} \mid \varnothing]\!] \\
\left(\begin{array}{l}
\mathit{done\_mission}\,.\,\mathit{currentMission} \longrightarrow \\
\mathit{end\_termination}\,.\,\mathit{sequencer} \longrightarrow \mathbf{Skip}
\end{array}\right)
\end{array}\right)
\end{array}\right) \\
\quad \square\ \mathit{currentMission} = \mathit{nullMissionId} \longrightarrow \\
\qquad \mathit{terminating} := \mathit{TRUE}\ ;\ \mathbf{Skip} \\
\quad \mathbf{fi}
\end{array}
$$

Figure 4.11: The Original *TopLevelMissionSequencerFW*'s *StartMission* Action

necessary (indicated by its model having far more states than the current protocol). Indeed, it was the process of formally modelling SCJ Level 2 that first illuminated the complexities of the mission sequencer termination protocol. These complications only became apparent at Level 2 because of its capacity to nest mission sequencers arbitrarily deeply, which means that mission sequencers could be terminated by schedulables both above and below themselves in the program's hierarchy at any point during the execution phase.

4.5.1 Model of the Original Termination Protocol

The original termination protocol required both the *TopLevelMissionSequencerFW* and the *SchedulableMissionSequencerFW* to be very complex process because mission sequencers could be terminated at an arbitrary point during their execution. In this section we describe our model of the original protocol and explain the source of its complexity. We present the model of the top-level mission sequencer, the model of the schedulable mission sequencer, the model of the `requestSequenceTermination()` method (from the original protocol), and the model the mission's `cleanUp()` method.

Top-Level Mission Sequencer

The *TopLevelMissionSequencerFW* process has one parameter, *sequencer*, which is the identifier of this mission sequencer process, and two variables: *currentMission*, which holds the identifier of this mission sequencer's current mission; and *terminating*, which is a boolean value that records if this mission sequencer has been asked to terminate. The *GetNextMission* action models the `getNextMission()` method and is shown in Fig. 4.10.

It communicates with the application model using the channels *getNextMissionCall* and *getNextMissionRet* to get the identifier of the next mission that this mission sequencer should execute. This identifier is stored in the variable *currentMission*. Then the *StartMission* action (Fig. 4.11) is called; it uses the *start_mission* channel to start the current mission.

Once the current mission enters its execution phase (indicated by the communication on

```
if $terminatingAbove = FALSE \land terminatingBelow = FALSE \longrightarrow$
  $GetNextMission$
[] $terminatingAbove = TRUE \lor terminatingBelow = TRUE \longrightarrow$
  **Skip**
```

Figure 4.12: Part of the Original *SchedulableMissionSequencerFW*'s *GetNextMission* Action

the *initializeRet* channel) the *RequestSequenceTermination* action is offered in parallel with a communication on the *done_mission* channel, which is used by the current mission to indicate that it has terminated.

The parallelism here specifies that *RequestSequenceTermination* and the communications in the brackets below the parallel operator synchronise on *end_termination*, that *RequestSequenceTermination* alters the *terminating* variable, and the behaviours below the parallel operator do not alter any variables.

Once a communication on *done_mission* occurs, the *StartMission* action waits for the *RequestSequenceTermination* action to be ready to engage in *end_termination*; this ends both sides of the parallelism and control returns to the *GetNextMission* action. Then, the *GetNextMission* action checks the value of the variable *terminating*, which is set in the *RequestSequenceTermination* action (shown in Fig. 4.15 and discussed below) to determine whether it should recurse or exit.

Schedulable Mission Sequencer

The *SchedulableMissionSequencerFW* process represents a schedulable mission sequencer; it is slightly more complicated than the *TopLevelMissionSequencerFW*. It has a parameter, *sequencer*, and a variable, *currentMission*, like the top-level mission sequencer process. However, instead of the *terminating* variable it has two state components: *terminatingAbove*, which indicates if this mission sequencer's controlling mission has asked it to terminate, and *terminatingBelow*, which indicates if this mission sequencer's current mission (or one of its schedulables) has asked it to terminate. Two variables are required because we have to treat the requests for termination differently, depending on their source. With two variables, we can model the different protocols separately.

The *GetNextMission* action models the `getNextMission()` method, and behaves the same as that in the *TopLevelMissionSequencerFW* process (Fig. 4.10) with the addition of the conditional statement shown in Fig. 4.12. This conditional checks both *terminatingAbove* and *terminatingBelow* to handle the possibility of the schedulable mission sequencer being asked to terminate from above or below itself in the program hierarchy.

The *StartMission* action shown in Fig. 4.13, contains a parallelism of three actions that offer the choice of waiting for its controlling mission to signal its termination (handled by the *SignalTermination* action), the *RequestSequenceTermination* action (which we discuss below), and waiting for its current mission to communicate its termination on *done_mission*.

The *SignalTermination* action in Fig. 4.14 handles the schedulable mission sequencer

$$
\begin{array}{l}
StartMission \mathrel{\widehat=} \\
\quad \mathbf{if}\ currentMission \neq nullMissionId \longrightarrow \\
\qquad \left(\begin{array}{l}
start\_mission \,.\, currentMission \longrightarrow \\
initializeRet \,.\, currentMission \longrightarrow \\
\left(\begin{array}{l}
\left(\begin{array}{l}
SignalTermination \\
\quad \llbracket \{ terminatingAbove \} \mid \{\!| end\_terminations |\!\} \mid terminatingBelow \rrbracket \\
RequestSequenceTermination
\end{array}\right) \\
\quad \llbracket \{ terminatingAbove, terminatingBelow \} \mid \{\!| end\_terminations |\!\} \mid \varnothing \rrbracket \\
\left(\begin{array}{l}
done\_mission \,.\, currentMission \longrightarrow \\
end\_terminations \,.\, sequencer \longrightarrow \mathbf{Skip}
\end{array}\right)
\end{array}\right)
\end{array}\right) \\
\quad [\!]\ currentMission = nullMissionId \longrightarrow \\
\qquad terminating := TRUE \\
\quad \mathbf{fi}
\end{array}
$$

Figure 4.13: Original *SchedulableMissionSequencerFW*'s *StartMission* Action

$$
\begin{array}{l}
SignalTermination \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\left(end\_terminations \,.\, sequencer \longrightarrow \mathbf{Skip} \right) \\
\Box \\
\left(\begin{array}{l}
signalTerminationCall \,.\, sequencer \longrightarrow \\
terminatingAbove := TRUE; \\
requestTermination \,.\, currentMission \longrightarrow \\
signalTerminationRet \,.\, sequencer \longrightarrow \mathbf{Skip}
\end{array}\right) \\
;\ end\_terminations \,.\, sequencer \longrightarrow \mathbf{Skip}
\end{array}\right)
\end{array}
$$

Figure 4.14: The Original *SchedulableMissionSequencerFW*'s *SignalTermination* Action

being terminated from above and the *done_mission* communication handles its current mission telling it to terminate from below. The *RequestSequenceTermination* action handles the schedulable mission sequencer being told to terminate its sequence of missions by a managed schedule. We discuss this action next.

Request Sequence Termination

The *RequestSequenceTermination* action, shown in Fig. 4.15, waits for a communication on the *requestSequenceTermination* channel. After this, the value of *terminating* is set to *TRUE* and the mission is queried to see if it is active and has not been asked to terminate already – using the channels *terminationPending* and *missionActive*. If these conditions are met, the action communicates on *requestTermination*, which tells the current mission to begin terminating. Then, *RequestSequenceTermination* recurses, so that subsequent calls to `requestSequenceTermination()` in the SCJ application can be handled, and so that the action can be terminated using *end_termination*.

$$
\begin{array}{l}
\mathit{RequestSequenceTermination} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{requestSequenceTermination} \,.\, \mathit{sequencer} \longrightarrow \\
\mathit{terminating} := \mathit{TRUE}; \\
\mathit{terminationPending} \,.\, \mathit{currentMission} \,?\, \mathit{missionTerminating} \longrightarrow \\
\mathit{missionActive} \,.\, \mathit{currentMission} \,?\, \mathit{missionIsActive} \longrightarrow \\
\left(\begin{array}{l}
\mathbf{if}\ \mathit{missionTerminating} = \mathit{FALSE} \land \mathit{missionIsActive} = \mathit{TRUE} \longrightarrow \\
\quad \mathit{requestTermination} \,.\, \mathit{currentMission} \longrightarrow \\
\quad \mathbf{Skip} \\
[\!]\ \mathit{missionTerminating} = \mathit{TRUE} \lor \mathit{missionIsActive} = \mathit{FALSE} \longrightarrow \\
\quad \mathbf{Skip} \\
\mathbf{fi}
\end{array} \right) \\
;\ \mathit{RequestSequenceTermination}
\end{array} \right) \\
\Box \\
\left(\mathit{end\_termination} \,.\, \mathit{sequencer} \longrightarrow \mathbf{Skip} \right)
\end{array} \right)
\end{array}
$$

Figure 4.15: The Original *RequestSequenceTermination* Action

$$
\mathit{CleanupSchedulables} \mathrel{\widehat=} \left(\begin{array}{l}
\interleave\ s : \mathit{schedulables} \bullet \\
\quad \mathit{cleanupSchedulableCall} \,.\, s \longrightarrow \\
\quad \mathit{cleanupSchedulableRet} \,.\, s \longrightarrow \mathbf{Skip}
\end{array} \right)
$$

Figure 4.16: sOriginal *Mission*'s *CleanupSchedulables* Action

In the *SchedulableMissionSequencerFW* process, the *RequestSequenceTermination* action differs only in that, where *terminating* is set to *TRUE*, the variable *terminatingBelow* is set instead. This is to handle the schedulable mission sequencer being terminated from a schedulable that is above it in the program hierarchy using *SignalTermination*, or below it, using *RequestSequenceTermination*. This can be seen in Fig. 4.15, where *SignalTermination* sets *terminatingAbove*, and in the excerpt of *GetNextMission* in Fig. 4.12, which checks both *terminatingAbove* and *terminatingBelow*.

Clean Up

Our model of a mission uses three actions to model its three phases of operation: initialisation, execution, and clean up. As soon as one phase ends, the mission transitions to the next phase. Hence, the mission's *Cleanup* action begins directly after its *Execute* action has finished. First, the *CleanupSchedulables* action is called (Fig. 4.16), which iterates over the set *schedulables* and triggers the *cleanUp*() action of each of the schedulables, calling the *cleanupSchedulableCall* event followed by the *cleanupSchedulableRet* event. The interleave operator ($\interleave$) is used to interleave each schedulable's clean up phase action.

Once the clean up of each of this mission's registered schedulables has finished, the *Cleanup* action executes the mission's cleanup behaviour, using the *cleanupMissionCall* and

$$\begin{array}{l} Finish \mathrel{\widehat=} \\ \quad end\_mission\_app\,.\,mission \longrightarrow \\ \quad done\_mission\,.\,mission \longrightarrow \\ \quad \mathbf{Skip} \end{array}$$

Figure 4.17: Original *Mission*'s *Finish* Action

cleanupMissionRet events. Afterwards, the *Finish* action (Fig. 4.17) is executed; it informs the mission's application process to terminate (*end_mission_app*) and then informs the mission's controlling mission sequencer that is has finished (*done_mission*).

This model captures the original termination protocol. While the model is tractable, we argue that the same functionality can be achieved with a simpler termination protocol. In Sect. 4.5.2 we describe our model of the termination protocol including our proposed changes.

4.5.2 Model of Current Termination Protocol

This section describes our model of the current SCJ termination protocol, which incorporates that changes that we proposed in [56]. In the current protocol there are two changes from the protocol described in Sect. 4.5.1. First, the `requestSequenceTermination()` method is removed, which prevents the arbitrary termination of mission sequencers. Second, the `Mission.cleanUp()` method is altered so that it returns a boolean, which indicates if this mission's controlling mission sequencer should continue or terminate. This enforces a hierarchical termination of mission sequencers.

In adapting the original model to the current protocol, the state components of both flavours of mission sequencer process were altered. In the *TopLevelMissionSequencerFW* process (Appendix C.9), the variable *terminating* was replaced with *continue*; and in the *SchedulableMissionSequencerFW* process (Appendix C.11), the variables *terminatingAbove* and *terminatingBelow* were replaced with *continueAbove* and *continueBelow*. These variables indicate to the sequencer that it should continue executing its sequence of missions (if they are both *TRUE*).

If *continue* is *FALSE* (or either *continueAbove* or *continueBelow* is *FALSE* in the case of the *SchedulableMissionSequencerFW*) then the mission sequencer does not execute any more missions. In the *SchedulableMissionSequencerFW* process the variable *continueBelow* holds the return value from the current mission that is communicated to the mission sequencer at the end of the cleanup phase (on the *done_mission* channel); this value is held in the *continue* variable in the *TopLevelMissionSequencerFW* process. The *continueAbove* variable, in the, *SchedulableMissionSequencerFW*, is only changed during the *SignalTermination* action, which handles the mission sequencer's controlling mission requesting it to terminate.

Removing the *RequestSequenceTermination* action is a clear simplification of the model; the *requestSequenceTermination* channel is no longer needed and is removed from the model entirely. Besides this, the actions (in the model of the original termination protocol) that use the *RequestSequenceTermination* action are also simplified in the model of the current protocol. We give more details of these simplifications in the following three sections. We

```
$StartMission \mathrel{\widehat=}$
  $\mathbf{if}\ currentMission! = nullMissionId \longrightarrow$
    $\left(\begin{array}{l} start\_mission\ .\ currentMission \longrightarrow \\ done\_mission\ .\ currentMission\ ?\ continueReturn \longrightarrow \\ continue := continueReturn\ ;\ \mathbf{Skip} \end{array}\right)$
  $[]\ currentMission = nullMissionId \longrightarrow$
    $continue := FALSE$
  $\mathbf{fi}$
```

Figure 4.18: Current *TopLevelMissionSequencerFW*'s *StartMission* Action

present the models of the top-level mission sequencer, the schedulable mission sequencer, and the mission's cleanup method.

Top-Level Mission Sequencer

The *StartMission* action in the *TopLevelMissionSequencerFW* process is simpler than the previous version (in Fig. 4.11), as can be seen from the excerpt presented in Fig. 4.18.

This action simply starts the current mission using the *start_mission* channel and then waits for it to terminate and communicate on the *done_mission* channel. The parallelism (executing *RequestSequenceTermination*) in the previous version of this action has been removed.

Schedulable Mission Sequencer

The *StartMission* action in the *SchedulableMissionSequencerFW* process (which models a schedulable mission sequencer) is shown in Fig. 4.19. It is needfully more complex than the same action in the *TopLevelMissionSequencerFW* process, but it is still simpler than in the model of the original protocol in Fig. 4.13.

After the mission has been initialised (indicated by the *initializeRet* channel) this action proceeds to a parallelism that offers *SignalTermination* to handle this mission sequencer's controlling mission being terminated and a communication on *done_mission* that indicates that the mission sequencer's current mission has terminated. This has eliminated one parallel process (*RequestSequenceTermination*) that was presenting in the previous version of this action.

Clean Up

To model `Mission.cleanUp()`, which now returns a boolean value, the *MissionFW* process's *cleanupMissionRet* channel takes a boolean parameter, *continueSequencer*, which is passed to the *Finish* action. This can be seen in the *Cleanup* action, shown in Fig. 4.20.

The value of *continueSequencer* is communicated to the *MissionSequencer* process via the *done_mission* channel, shown in the *Finish* action in Fig. 4.21. This channel is the means

$$
\begin{array}{l}
StartMission \mathrel{\widehat=} \\
\quad \mathbf{if}\ currentMission ! = nullMissionId \longrightarrow \\
\qquad \left(\begin{array}{l}
start\_mission \,.\, currentMission \longrightarrow \\
initializeRet \,.\, currentMission \longrightarrow \\
\left(\begin{array}{l}
SignalTermination \\
\quad [\![\varnothing \mid \{\!| end\_terminations |\!\} \mid \{ continueBelow \}]\!] \\
\left(\begin{array}{l}
done\_mission \,.\, currentMission \,?\, continueReturn \longrightarrow \\
continueBelow := continueReturn; \\
end\_terminations \longrightarrow \mathbf{Skip}
\end{array}\right)
\end{array}\right)
\end{array}\right) \\
\quad [\!]\ currentMission = nullMissionId \longrightarrow \\
\qquad continueBelow := FALSE \\
\quad \mathbf{fi}
\end{array}
$$

Figure 4.19: Current *SchedulableMissionSequencerFW*'s *StartMission* Action

$$
\begin{array}{l}
Cleanup \mathrel{\widehat=} \\
\quad CleanupSchedulables; \\
\quad cleanupMissionCall \,.\, mission \longrightarrow \\
\quad cleanupMissionRet \,.\, mission \,?\, continueSequencer \longrightarrow \\
\quad Finish(continueSequencer)
\end{array}
$$

Figure 4.20: Current *Cleanup* Action

of communication that allows a mission to inform its controlling mission sequencer of its completion, and, as revised, also communicates this continuation information.

When the *MissionSequencer* receives the boolean value from *done_mission*, it stores it in the variable *continue*, which is checked by the *GetNextMission* action after the *StartMission* action finishes. This variable is used to decide whether the *MissionSequencer* should continue its execution and get another mission, or terminate. This minor addition to the model presents little extra complexity, while supporting our proposal to simplify the termination protocol significantly. Section 4.5.3 compares the two termination protocols in more detail.

4.5.3 Comparison of Termination Protocols

The original termination protocol allowed any schedulable to call any mission sequencer's `requestSequenceTermination()` method, regardless of its place in the hierarchy. The *Circus* action modelling this method is presented in Fig. 4.15. The mission sequencer that receives this call informs its current mission to terminate. This is shown in Fig. 4.15 by the communication on the *requestTermination* channel, which indicates to the mission that it should terminate. The mission, once it is instructed to terminate, passes this on to its schedulables – at least one of which may have called the `requestSequenceTermination()` method of the mission sequencer in the first place. This created a needless cycle of termination requests.

$$
\begin{array}{l}
\mathit{Finish} \mathrel{\widehat=} \mathbf{val}\, \mathit{continueSequencer} : \mathbb{B} \bullet \\
\quad \mathit{end\_mission\_app} \,.\, \mathit{mission} \longrightarrow \\
\quad \mathit{done\_mission} \,.\, \mathit{mission}\, !\, \mathit{continueSequencer} \longrightarrow \\
\quad \mathbf{Skip}
\end{array}
$$

Figure 4.21: Current *Finish* Action

In the current termination protocol, which we proposed in [56], the instigation of termination still begins in a schedulable object, but this request is only passed up one tier at a time. For example, if a reason to terminate the application is detected by a schedulable, this is passed to its controlling mission – by setting some flag in the mission, for example. During the mission's cleanup phase, it communicates this request for termination to its controlling mission sequencer. This is captured in our models by the communication on the *done_mission* channel of the boolean parameter *continueSequencer* to the particular mission sequencer process that controls the mission. In this way, the request for termination passes up the program hierarchy, with each tier terminating before the next tier begins handling its termination.

This prevents the situation in the original protocol in which a schedulable initiates the termination of a sequence of missions that includes its controlling mission, and then waits to be terminated itself when its controlling mission is terminated. The current approach does create a small amount of programmer overhead, since the programmer must ensure that schedulables can inform their controlling mission that it should return `false` from its `cleanUp()` method. A simple way to remove this small overhead is for the default implementation of `Mission.cleanUp()` to return `false`. We note that even in the current termination protocol, the schedulable that discovers the need for termination triggers the termination of its controlling mission and is then asked to terminate itself. To avoid this, the schedulables can be programmed to check for the termination of their controlling mission periodically and begin to shut themselves down; this is in fact the only way to terminate a `ManagedThread`. Another solution is to have the schedulable that discovers the need for termination terminate itself after it has triggered the termination of its controlling mission.

Our proposal has a subtle effect on the termination order of the objects in the program. As an example, we consider a program with two nested subsystems. With the original protocol, a schedulable within one of them may call `requestSequenceTermination()` on the top-level mission sequencer and begin a cascade of termination requests that leads to the nested sub-systems terminating in parallel. In the same situation, using the current termination protocol, the termination requests must pass up the hierarchy from the schedulable that initiates termination to the top-level mission sequencer. This means that the subsystem that contains the schedulable that requested termination has to terminate before the request for termination passes to the top-level mission and the termination of the other subsystem – and any schedulables started by the top-level mission – begins.

In summary, the `requestSequenceTermination()` method complicated the SCJ termination protocol by allowing arbitrary termination of mission sequencers. Our models, while tractable, were complex when modelling this feature of SCJ at Level 2. With our proposed changes incorporated, our models became much simpler and easier to analyse. Our proposed changes to the SCJ termination protocol represent a positive simplification of the language while retaining the ability to terminate a mission sequencer from the application.

In order to show how far our proposed changes simplify the model of SCJ, we constructed two specifications of a simple program similar to that presented in Sect. 2.1.2, which contains a single mission, controlling two managed threads that share a one-place buffer in the mission's memory. One specification uses the model of the original framework and the other specification uses the model of the current framework, which incorporates our proposal. Both of these specifications were translated to CSP_M, the machine-readable version of CSP, in order to utilise FDR3 [33] to model check the specifications. Note that, because CSP does not have a notion of state in the same way that *Circus* does, the CSP versions of our models use state processes to control variables instead, which means that the CSP models have more states than the *Circus* versions.

The results obtained are from running a check for divergence-freedom while hiding any channels relating to the state processes. The model of the original framework shows 4,539,021 states, whereas the model of the current framework shows only 249,869 states. Our proposed changes decreased the number of states in the model by 94.5% in comparison to the original. We note that the number of states here is higher than the number of states often found in models of full programs because these checks only use the framework model, which will often exhibit more states when not driven by an application model. This decrease in the number of states in our model shows a simplification that is useful for further modelling efforts and for programmer understanding of the SCJ paradigm.

4.6 Summary

This chapter presents the first formal model of SCJ Level 2, written in the state-rich process algebra *Circus*. The full model can be found in Appendix C. We follow the approach used to model SCJ Level 1 [93]. Our model captures the state and behaviour of the Level 2 paradigm, exceptions (where they indicate a misuse of the paradigm), inheritence and polymorphism, and, finally, synchronisation and suspension.

We do not simply add Level 2 features to the Level 1 model; our model captures API changes that occurred after the Level 1 model was constructed and some features of Level 1 that were omitted from the original model for brevity. For example, mutually exclusive synchronised methods are available at Level 1, but they are not captured by the Level 1 model. Our Level 2 model captures this feature. Further, the Level 2 model captures the release patterns of event handlers, so that this does not have to be captured during program translation, and considers that deadlines can be overrun.

This model provides both top-down and bottom-up benefits. Top-down, it is a target for the *Circus* refinement strategy [17] to develop SCJ Level 2 programs that are correct by construction. Bottom-up, it can be used as a program verification tool via a translation of the model to CSP_M for analysis using FDR3.

The modelling process itself has also proved beneficial. During modelling we found that the original SCJ termination protocol was more complicated than needed and used our model to show that a proposed protocol (which is the protocol currently used by SCJ programs) was simpler, while allowing the same functionality.

In the next chapter, we describe the process of translating a given SCJ Level 2 program into our model. The chapter shows a detailed, step-by-step process for translating programs; describes a formalisation of the core elements of the translation, written in Z; and describes a tool to automate the translation, called $\text{T}^{\underline{\text{ight}}}\text{R}^{\underline{\text{ope}}}$, which is written in Java.

Chapter 5

SCJ Level 2 Translation

In this chapter we present a translation strategy that captures the application-specific information from a SCJ Level 2 program and produces a model of this behaviour that is compatible with the model of the SCJ API presented in Chap. 4. Section 5.1 presents this translation strategy in a step-by-step manner. This is presented first to better inform the reader for the discussion that follows. We use the Buffer example, in Sect. 2.1.2, to illustrate the translation. Section 5.2 presents a formalisation in Z of the core elements of the translation strategy. Section 5.3 describes $\mathrm{T}^{\mathrm{ight}}\mathrm{R}^{\underline{\mathrm{ope}}}$, a prototype tool that automatically translates SCJ Level 2 programs into our *Circus* model. Section 5.4 evaluates the translation work presented in this chapter. Finally, Sect. 5.5, summarises the translation strategy, core formalisation, and automatic translation tool presented in this chapter.

5.1 Translation Strategy

Our translation strategy captures the application-specific information in an SCJ Level 2 program and produces the application model of its behaviour. This comprises the models of each object in the program and processes that control the network of those models. The generic behaviour of SCJ programs is already captured by the framework model, described in Chap. 4. Combining these two models forms the model of a program. This approach simplifies the translation and the models that it produces.

Paradigm objects (the safelet, mission sequencers, missions, and schedulables) each have a template that describes the structure required by each type of process. Non-paradigm objects do not have framework models; they are, by definition, application specific. Therefore, they all share a simpler template that captures the variables and methods of the class.

Where objects in the program contain non-reactive behaviour, for example methods that only perform data operations, they are represented by an extra component. As an example of this. we consider the `Buffer` class from the buffer example in Sect. 2.1.2. The `Buffer` class is non-reactive, as it simply holds an integer variable and provides methods to access and update it. This sort of non-reactive behaviour is represented by an *OhCircus* class. Objects that contain a mixture of reactive and non-reactive behaviour are represented by both a *Circus* process and *OhCircus*, which cooperate to model the object.

SCJ programs are arranged hierarchically. The safelet is at the top of the hierarchy and controls the program. It loads a top-level mission sequencer, which loads a sequence of

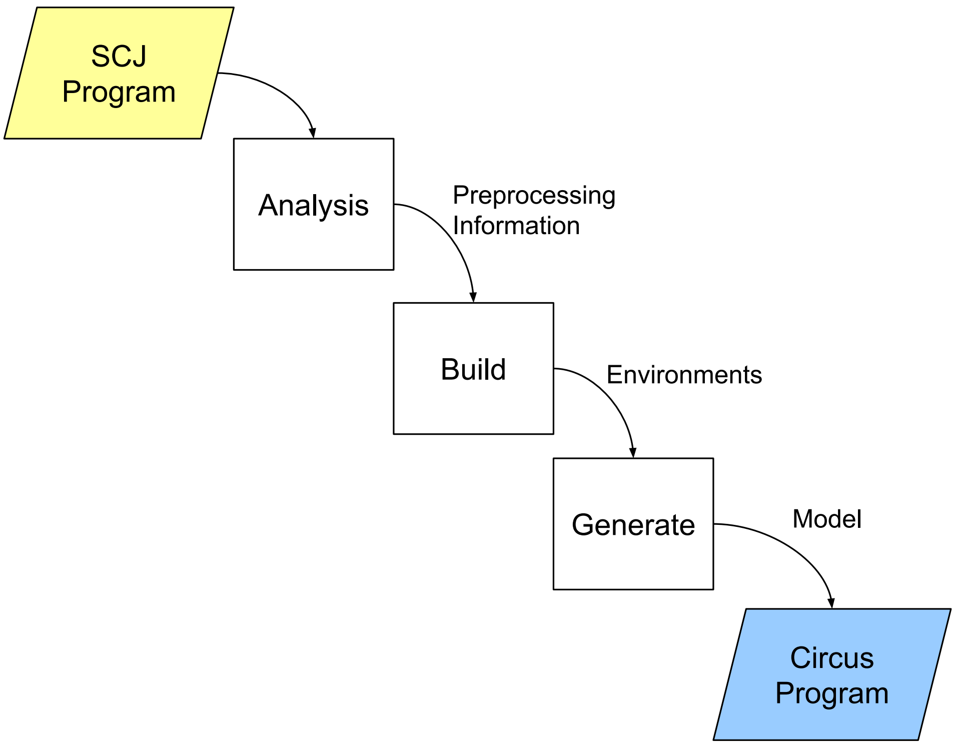


Figure 5.1: Flow Diagram of the Translation Phases

missions. The safelet and top-level mission sequencer compose the *Control Tier* of a program. Note that while a program will only have one top-level mission sequencer while it is running, the `getSequencer()` method may return one of several mission sequencers. The rest of the program is organised into *Program Tiers*. A program tier is a list of *Cluster*s, which each contain a mission and its registered schedulables.

The clusters containing the missions loaded by the top-level mission sequencer comprise Tier0; all programs will have a Tier0. Subsequent program tiers indicate missions in Tier0 registering a schedulable mission sequencer. We characterise the structure of programs this way to make it easier to construct the model of a program at the top level. The arrangement of the tiers and clusters is captured by a specific template.

The translation occurs in three phases, shown in Fig. 5.1. In the next three sections we describe each phase of the translation and illustrate the information that is produced. We use the `MainMission` class from the `Buffer` example, shown in Fig. 5.2, as a running example. First, the analysis phase (Sect. 5.1.1) identifies the program's object types, methods and their locations, and the callers of the methods. Second, the build phase (Sect. 5.1.2) extracts information from the SCJ program and constructs an environment for each object and the program as a whole. Third, the generate phase (Sect. 5.1.3) produces the model from the data in the environments.

5.1.1 Analysis

The analysis phase produces three maps that contain information about the program, to help the other two phases of the translation, and is achieved by two steps. The first step builds a map of each object to the template it requires. Table 5.1 shows the marker that identifies each type of object and the template that should be selected. There are eight application process

```
public class MainMission extends Mission
{
  private final Buffer buffer;

  public PCMission(){
    buffer = new Buffer();
  }

  protected void initialize(){
    StorageParameters storageParameters = new StorageParameters(150 * 1000,
        Const.PRIVATE_MEM_DEFAULT, Const.IMMORTAL_MEM_DEFAULT,
        Const.MISSION_MEM_DEFAULT - 100 * 1000);

    new Producer(new PriorityParameters(10), storageParameters, this).register();

    new Consumer(new PriorityParameters(10), storageParameters, this).register();
  }

  public Buffer getBuffer(){
    return buffer;
  }

  public boolean cleanUp()  {
    return false;
  }
}
```

Figure 5.2: The `MainMission` Class from the `Buffer` Example

template, one for each type of the paradigm objects and one for non-paradigm objects. These templates define the structure of the application processes. The application model of each paradigm object has an action for each of its overridden API methods. For example, the application model of a mission has actions capturing the application-specific behaviour of the `initialize()` and `getNextMission()` methods, respectively. These actions are triggered by the framework model when it encounters application-specific behaviour. Additionally, the application models capture a paradigm object's application-defined variables and methods.

Since each paradigm object extends a different class (or implements an interface, in the case of the safelet) and requires a different output template, it is simple to identify the output template of each paradigm object. The only complication is that the template type of a mission sequencer cannot be identified until the build phase (in Sect. 5.1.2). This is because we model a mission sequencer that is started by a safelet differently to a mission sequencer that is started by a mission, as described in Sect. 4.2.2. Since this cannot be ascertained without checking the contents of a method, we defer this until the build phase. Non-paradigm objects – any object of a class that is not a safelet, mission sequencer, mission, or schedulable – share an output template, so this mapping is also simple to identify.

The second step produces two maps that are used in the generate phase to produce the *MethodCallBinder* process. The first is the Class Method Map, which maps each class to the names of its methods that are not defined in the SCJ API. The second is the Method Callers Map, which maps each application-defined method in the program to the names of the classes that call it. To deal with name clashes, two methods are considered to be the

Object type	Marker	Template Type
Safelet	`implments Safelet`	SafeletApp
Mission Sequencer	`extends MissionSequencer`	TopLevelMissionSequencerApp <u>or</u>
		SchedulableMissionSequencerApp
Mission	`extends Mission`	MissionApp
Periodic Event Handler	`extends PeriodicEventHandler`	PeridoicEventHandlerApp
Aperiodic Event Handler	`extends AperiodicEventHandler`	AperiodicEventHandlerApp
One-Shot Event Handler	`extends OneShotEventHandler`	OneShotEventHandlerApp
Managed Thread	`extends ManagedThread`	ManagedThreadApp
Non-Paradigm Object	None of the above	NonParadigmApp

Table 5.1: Object Types, their Markers and Templates

same if their names are the same and they are declared in the same class or classes related by inheritance. If two methods share only the same name, but are not declared in classes related by inheritance, then the method name is prefixed by the class name to produce unique action and channel names.

As a example of the output from the analysis phase we consider the `MainMission`, shown in Fig. 5.2, from the buffer example (described in Sect. 2.1.2). The first step of the analysis phase identifies `MainMission` as a mission, because it extends the `Mission` class. Step two analyses each of the methods in `MainMission` and finds that only `getBuffer()` is not defined by the SCJ API. This means that the second step adds a mapping between `MainMission` and a list containing `getBuffer` (`MainMission` $\mapsto$ `[getBuffer]`) to the Method Class Map. As the `MainMission` does not call any application-defined methods, the second step does not update the Method Callers Map.

5.1.2 Build

The build phase takes the SCJ program and the maps produced by the analysis phase (Sect. 5.1.1), performs a translation of elements of the program, and outputs an environment for each object. For each type of object – safelet, mission sequencer, mission, one of the schedulables, or non-paradigm object – this phase builds a different environment. The environment representing a class contains its name, variables, and methods. Each variable and method is also represented by an environment, which contains its translation.

The build phase also constructs an overall environment of the program. This environment holds information about all of the objects in the program and about its structure. This is passed on to the generate phase to produce the part of the model that controls the network of processes.

In the rest of this section we present an overview of the build phase. This comprises a description of the process that builds an environment (which is reused and extended when building the environments for each type of object), and then, in order, the process of building: non-paradigm objects, safelet, mission sequencers, missions, and schedulables.

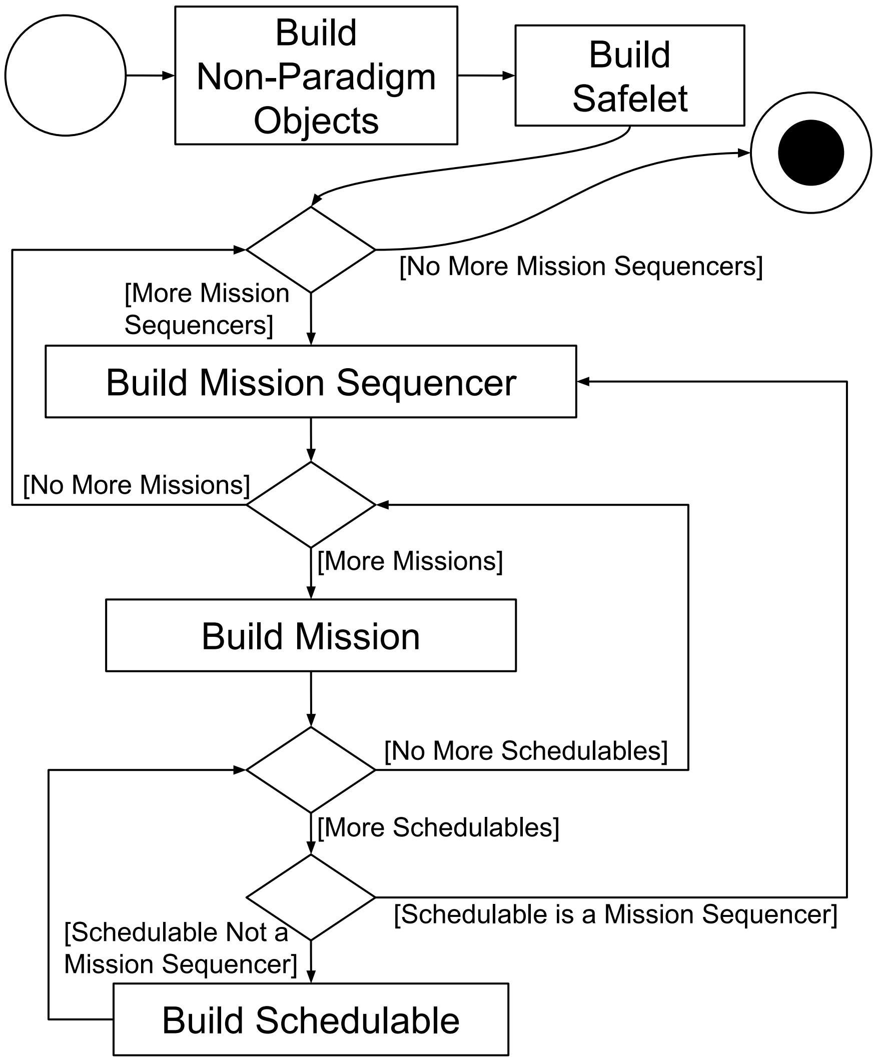

Build Non-Paradigm Objects
Build Safelet
[No More Mission Sequencers]
[More Mission Sequencers]
Build Mission Sequencer
[No More Missions]
[More Missions]
Build Mission
[No More Schedulables]
[More Schedulables]
[Schedulable is a Mission Sequencer]
[Schedulable Not a Mission Sequencer]
Build Schedulable

Figure 5.3: Flow Chart of the Processes in the Build Phase

Overview

Figure 5.3 illustrates the build phase. The non-paradigm objects are the first to be translated, because they are unrelated to the SCJ program's structure. Then the paradigm objects are translated. Because they form a hierarchy, the translation strategy uses the structure of the program to inform the order of translation, and branches to ensure that every paradigm object is translated. Each type of object in the program has a different process to build its environment. However, because the environments share a common definition of the information of a class, they share a sub-process that builds the common elements of those environments.

The safelet identified by the analysis phase (Sect. 5.1.1), is the first paradigm object to be translated because it is the component at the top of the program hierarchy. We build the environment of the safelet and then retrieve the mission sequencers that may be returned by its `getSequencer()` method. For each of these top-level mission sequencers, we build all of the environments of the missions that can be returned by its `getNextMission()` method. For each mission, we build the environments of its schedulables until there are none left. Once the environments of all of the missions from a particular mission sequencer have been built, we build the environment of the next mission sequencer. Because we build the tree of an SCJ program depth-first, if there are no more top-level mission sequencers to build then we have explored the whole program and we can proceed to the generate phase, described in Sect. 5.1.3.

Figure 5.4 shows an EBNF description of the environments produced by the build phase. The *NetworkEnv* category defines the network environment, which holds information for producing the network of processes that control the model at the top level. It contains the *ProgramEnv*, which we describe next; the *AppEnv*, which contains a list of the names and parameters of the application processes; and the *LockingEnv* containing the names and priorities of the *ThreadFW*s and the names of the *ObjectFW* processes. The *LockingEnv* may be empty if the program does not use synchronisation or suspension.

The *ProgramEnv* category defines the program environment, which contains the class environments and characterises the structure of the program being translated. It comprises the *SafeletEnv*, an optional list of *MissionSequencerEnv*s, a list of *TierEnv*s, and a number of *ClassEnv*s. The *TierEnv* comprises a list of *ClusterEnv*s, which each contain a *MissionEnv* and a list of *SchedulableEnv*s representing its schedulables. This structure mirrors that of the network of processes that characterise the framework processes at the top level. Finally, the list of *ClassEnv*s represents the non-paradigm objects in the program.

A *ClassEnv* contains information extracted from a class in the SCJ program. Each of the environments that represent paradigm objects is defined as a *ClassEnv*, but three of them contain an extra list that holds structural information used to guide the translation. The *SafeletEnv* holds a list of the names of the mission sequencers returned from the safelet's `getSequencer()` method. A *MissionSequencerEnv* holds a list of the names of the missions returned from the mission sequencer's `getNextMission()` method. Finally, a *MissionEnv* environment holds a list of the names of the schedulables registered in the mission's `initialize()` method.

Building a Class Environment

As previously mentioned, the process of constructing a class's environment is reused by each build process in this phase. For each type of object, the build phase constructs a different environment. However, each of these environments is defined as a *ClassEnv* (some with extra information), so the process of extracting the information required by a *ClassEnv* is shared.

First, the name of the class is extracted and then translated (to ensure that it only contains characters that are valid in Z). Then the variables of the class are extracted. Here we are only concerned with the class variables (attributes), not variables declared in methods. For each class variable we build a *VarEnv* (defined in Fig. 5.4) that stores the translation of the name, type, and initialisation value of a variable. The *VarEnv* is added to the list of variable environments in the *ClassEnv*.

Then the class's methods are translated. For a class's constructors, their parameters (that are not strings or instances of the `ReleaseParameters`, `SchedulingParameters`, or `MemoryParameters` family of classes) become the parameters to the process. These are translated and stored in the *ClassEnv* The body of the constructor is analysed for variables assignments, which are used to update the initialisation of variables in the *ClassEnv*.

For each of a class's non-constructor methods we build a *MethodEnv* (Fig 5.4) and check if it is synchronised or not. If a method is synchronised then the *MethodEnv* representing it

$$
\begin{array}{lcl}
\mathit{NetworkEnv} & = & \mathit{ProgramEnv} \quad \mathit{AppEnv} \quad \mathit{LockingEnv} \\
\mathit{AppEnv} & = & \{\mathit{Name} \quad \mathit{Parameters}\} \\
\mathit{Threads} & = & \{\mathit{Name} \quad \mathit{PriorityLevel}\} \\
\mathit{Objects} & = & \{\mathit{Name}\} \\
\mathit{LockingEnv} & = & [\mathit{Threads}] \quad [\mathit{Objects}] \\
\mathit{ProgramEnv} & = & \begin{array}{l} \mathit{SafeletEnv} \quad [\mathit{MissionSequencerEnv}] \\ \{\mathit{TierEnv}\} \quad \{\mathit{ClassEnv}\} \end{array} \\
\mathit{TierEnv} & = & \{\mathit{ClusterEnv}\} \\
\mathit{ClusterEnv} & = & \mathit{MissionEnv} \quad \{\mathit{SchedulableEnv}\} \\
\mathit{PEHEnv} & = & \mathit{ClassEnv} \\
\mathit{APEHEnv} & = & \mathit{ClassEnv} \\
\mathit{OSEHEnv} & = & \mathit{ClassEnv} \\
\mathit{MTEnv} & = & \mathit{ClassEnv} \\
\mathit{SchedulableEnv} & = & \begin{array}{l} \mathit{MissionSequencerEnv} \\ \mid \mathit{PEHEnv} \\ \mid \mathit{APEHEnv} \\ \mid \mathit{OSEHEnv} \\ \mid \mathit{MTEnv} \end{array} \\
\mathit{MissionEnv} & = & \mathit{ClassEnv} \quad \mathit{Schedulables} \\
\mathit{Schedulables} & = & \{\mathit{Name}\} \\
\mathit{MissionSequencerEnv} & = & \mathit{ClassEnv} \quad \mathit{Missions} \\
\mathit{Missions} & = & \{\mathit{Name}\} \\
\mathit{SafeletEnv} & = & \mathit{ClassEnv} \quad \mathit{Sequencers} \\
\mathit{Sequencers} & = & \{\mathit{Name}\} \\
\mathit{ClassEnv} & = & \begin{array}{l} \mathit{Name} \quad [\mathit{Parameters}] \quad [\mathit{Variables}] \\ [\mathit{SyncMeths}] \quad [\mathit{Meths}] \quad [\mathit{ClassEnv}] \end{array} \\
\mathit{Meths} & = & \{\mathit{MethodEnv}\} \\
\mathit{SyncMeths} & = & \{\mathit{MethodEnv}\} \\
\mathit{Variables} & = & \{\mathit{VarEnv}\} \\
\mathit{Parameters} & = & \{\mathit{NameType}\} \\
\mathit{ReturnType} & = & \mathit{Type} \\
\mathit{MethodEnv} & = & \begin{array}{l} \mathit{Name} \quad \mathit{Parameters} \\ \mathit{ReturnType} \quad [\mathit{RetVal}] \quad \mathit{Body} \end{array} \\
\mathit{VarEnv} & = & \mathit{Name} \quad \mathit{Type} \quad \mathit{Init}
\end{array}
$$

Figure 5.4: EBNF Description of the Environments Produced by the Build Phase

Application	Result	Note
`boolean`	$\mathbb{B}$	where $\mathbb{B} ::= \mathbf{True} \mid \mathbf{False}$
`byte`	*byte*	where $byte = -128 \mathrel{..} 127$
`short`	*short*	where $short = -2^{15} \mathrel{..} 2^{15} - 1$
`int`	*int*	where $int = -2^{31} \mathrel{..} 2^{31} - 1$
`long`	*long*	where $long = -2^{63} \mathrel{..} 2^{63} - 1$
`float`	*float*	where *float* is a given type
`double`	*double*	where *double* is a given type
`ClassName`	`ClassName`*Class*	where `ClassName` is the name of a class or other reference type.

Table 5.2: Type Translation Rules

Application	Result
$\underline{\text{CVarDecl}}$; $\underline{\text{CVarDecls}}$	$CVarInit(\underline{\text{CVarDecl}})$ $\land$ $CvarInit(CVarDecl(\underline{\text{CVarDecls}}))$
$\underline{\text{AccesMod Type Name}}$;	$Name(\underline{\text{Name}}) = DefaultInit(\underline{\text{Type}})$
$\underline{\text{AccesMod Type Name}}$ = $\underline{\text{Expr}}$;	$Name(\underline{\text{Name}}) = Expr(\underline{\text{Expr}})$

Table 5.3: Class Variable Translation Rules

is recorded in the *ClassEnv*'s list of synchronised methods, if not, then it goes into the list of the non-synchronised methods.

For each method, we extract its name, its return type and potential values, and its parameters. We then translate the body of the method, capturing its behaviour. To achieve this we reuse translation rules from the Level 1 translation [93] because all three SCJ compliance levels are similarly restricted in their use of the Java language (except for suspension, which is only available at Level 2). Some alterations to these rules was required to accommodate the differences between our modelling approach and that of [93].

There are six categories that the EBNF in Fig. 5.4 does not define. The *PriorityLevel* category defines the range of priorities available to the program, which is defined in the SCJ API as being at least 1 . .28. The *Name* category describes the name of a class, method, or variable, so it can only include valid Z identifiers. The *Type* category describes a type, so it is a translation of a Java type into *Circus*, as defined in Table 5.2. The *Init* category describes the variable initialisation (defined in Table 5.3), so it is the translation of a value (a literal value, as described in Table 5.4 or the result of a statement, described in Table 5.5). The *ReturnValue* category describes the return value of a method, so it is also the translation of a value using the same rules as for *Init*. Finally, the *Body* category describes the body of a method, so it is the translation of a list of statements in the method body, which we describe in detail below.

To translate the body of a method we apply a translation rule to each statement. Each type of statement (for example, assignment, conditional, or return) has a rule to translate it into *Circus*. Table 5.5 describes the *Stmts* rule, which is used to translate most Java

Application	Result	Note
`0, 1, 2, ...`	$0, 1, 2, \ldots$	Unsigned Numbers
`0, -1, -2, ...`	$0, -1, -2, \ldots$	Signed Numbers
`true`	**True**	Boolean value true
`false`	**False**	Boolean value false
`null`	*null*	Null reference for data objects

Table 5.4: Literal Value Translation Rules

Application	Result
$\mathit{Stmts}\left(\underline{\text{Var}}\ \texttt{=}\ \underline{\text{Expr}}\ ;\right)$	$\mathit{Name}(\underline{\text{Var}}) := \mathit{Expr}(\underline{\text{Expr}})$
$\mathit{Stmts}\left(\underline{\text{Type}}\ \underline{\text{Var}}\ ;\ \underline{\text{Statements}}\right)$	$\textbf{var}\ \mathit{Name}(\underline{\text{Var}}) : \mathit{Type}(\underline{\text{Type}}) \bullet \mathit{Stmts}(\underline{\text{Statements}})$
$\mathit{Stmts}\left(\begin{array}{l}\texttt{if}\ (\underline{\text{Expr}})\ \underline{\text{IfBranch}}\\ \texttt{else}\ \underline{\text{ElseBranch}}\end{array}\right)$	$\textbf{if}(\ \mathit{Expr}(\underline{\text{Expr}})\ ) \longrightarrow \mathit{Stmts}(\underline{\text{IfBranch}})$ $[\!]\,(\neg\ \mathit{Expr}(\underline{\text{Expr}})\ ) \longrightarrow \mathit{Stmts}(\underline{\text{ElseBranch}})$
$\mathit{Stmts}\left(\begin{array}{l}\texttt{switch}\ (\ \underline{\text{Expr}}\ )\ \{\\ \texttt{case}\ \underline{\text{value}}\ :\ \underline{\text{Stmts}};\ \texttt{break;}\\ \ldots\\ \texttt{default}\ :\ \underline{\text{DefltStmts}}\ ;\ \}\end{array}\right)$	$\textbf{if}\ \mathit{Expr}(\underline{\text{Expr}}) = \mathit{Value}(\underline{\text{Value}}) \longrightarrow \mathit{Stmts}(\underline{\text{Stmts}})$ $[\!]\ \ldots$ $[\!]\,(\neg\ \mathit{Expr}(\underline{\text{Expr}}) = \mathit{Value}(\underline{\text{Value}}) \vee \ldots)$ $\longrightarrow \mathit{Stmts}(\underline{\text{DefltStmts}})$
$\mathit{Stmts}\left(\texttt{while}\ (\ \underline{\text{Expr}}\ )\ \{\ \underline{\text{Stmts}}\ \}\right)$	$\mu X \bullet \left(\begin{array}{l}\textbf{if}\ \mathit{Expr}(\underline{\text{Expr}}) \longrightarrow \mathit{Stmts}(\underline{\text{Stmts}})\ ;\ X\\ \ [\!]\neg\ \mathit{Expr}(\underline{\text{Expr}}) \longrightarrow \textbf{Skip}\\ \textbf{fi}\end{array}\right)$
$\mathit{Stmts}\left(\texttt{assert}\ \mathit{Expr}(\underline{\text{Expr}})\ ;\right)$	$\textbf{if}\ \mathit{Expr}(\underline{\text{Expr}}) \longrightarrow \textbf{Skip}$ $[\!]\ \neg\ \mathit{Expr}(\underline{\text{Expr}}) \longrightarrow \underline{\text{abort}}$ $\textbf{fi}$
$\mathit{Stmts}\left(;\right)$	$\textbf{Skip}$
$\mathit{Stmts}\left(\underline{\text{Stmts1}}\ ;\ \underline{\text{Stmts2}}\ ;\right)$	$\mathit{Stmts}(\underline{\text{Stmts1}})\ ;\ \mathit{Stmts}(\underline{\text{Stmts2}})$
$\mathit{Stmts}\left(\begin{array}{l}\texttt{for}\ (\ \underline{\text{Stmts1}}\ ;\ \underline{\text{Expr}}\ ;\ \underline{\text{Stmts2}}\ )\\ \underline{\text{ForBody}}\end{array}\right)$	$\left(\mu X \bullet \left(\begin{array}{l}\textbf{if}\ \mathit{Expr}(\underline{\text{Expr}}) \longrightarrow\\ \ \text{Stmts(ForBody)}\ ;\\ \ \text{Stmts(Stmts2)}\ ;\ X\\ \ [\!]\neg\ \mathit{Expr}(\underline{\text{Expr}}) \longrightarrow \textbf{Skip}\\ \textbf{fi}\end{array}\right)\right)$

Table 5.5: Statement Translation Rules

statements, except Java expressions, which are translated using the *Expr* rule in Table 5.6. These translation rules must often make use of other rules to translate specific elements of a statement or expression. The *Name* rule translates a name into a valid Z identifier. The *Type* rule translates a Java type into a valid Z type. Finally, the *Value* rule translates a Java value, literal or otherwise.

Translating method calls is more complex. The rules to translate method call statements are described in Table 5.7. A simple method call (`meth();`) implicitly targets `this` class (assuming that `meth()` does not belong to an imported class), whereas a more complex method call (`c.meth();`) targets the class `c`. However, the method may be located in the class that is that target of the call or inherited from a super class. We must also consider if a method is synchronised or not.

Each of the rules in Table 5.7 applies for a different type of method and method call. If the method is application-defined and is defined in the same class as the method invocation, or if the method is API-defined, then rule *Meth*1 is used, producing event names that simply combine the method name with *Call* and *Ret*, respectively. If the method is application-defined and is defined in another class, then rule *Meth*2 is used. This produces event names that combine '*binder_*' and the methods name. Such methods make use of the *MethodCallBinder* (*MCB*) process, described in Sect. 4.4, to bind the method invocation to its location.

Synchronised methods use rule *Meth*3, which includes an extra parameter to the call and return events. This parameter is the identifier of the *ThreadFW* process associated with the process making the method call. This uses the part of the framework model that handles synchronisation and suspension, described in Sect. 4.3. For all types of method call, method parameters are added as parameters to the *Call* event and return values are added as parameters to the *Ret* event.

As previously mentioned, we use *OhCircus* classes to capture non-reactive behaviour, for example methods that are purely data operations. The *ClassEnv* optionally holds a *CircClassEnv*, not shown in Fig. 5.4, which represents an *OhCircus* class that is associated with this *ClassEnv*; this models the situation where an object has both reactive and non-reactive behaviour, and so it captured by both a *Circus* process and an *OhCircus* class. *CircClassEnv* holds the same attributes as the *ClassEnv*, except for the *CircClassEnv* so as to avoid a circular definition.

As an example of an environment that is produced by the build phase, we return to the `MainMission` example shown in Fig. 5.2. The environment built for the `MainMission` class is shown in Table 5.8, where the values are given in a JSON-style format. The name is simply the name of the class: 'MainMission'. The class declares one variable, `Buffer buffer`, which declares no initialiser. `MainMission` has no synchronised methods, so the synchronised methods list in its environment is empty. Finally, the class has four non-synchronised methods. The class's constructor declares no parameters, so the parameters list in the environment is empty. The `buffer` variable is initialised in the constructor, so this is added to the environment for the `buffer` variable. The names, return values and types, and bodies of the `initialize()`, `cleanUp()`, and `getBuffer()` methods are extracted and translated. The `initialize()` method calls `register()` on two schedulables, a `Producer` and a `Consumer`. So the Schedulables list in the environment contains these two class names.

Application	Result	Note
Expr($\underline{\text{Value}}$)	*Value*($\underline{\text{Value}}$)	Where $\underline{\text{Value}}$ is a literal value
Expr($\underline{\text{UniOp Expr}}$)	*Expr*(UniOp) *Expr*(Expr)	Where $\underline{\text{UniOp}}$ is a unary operator
Expr($\underline{\text{Expr1 BinOp Expr2}}$)	*Expr*(Expr1) *Expr*(BinOp) *Expr*(Expr2)	Where $\underline{\text{BinOp}}$ is a binary operator
Expr(`+, -/ * ...`)	$+, -, *\ldots$	
Expr(`&&, ||, !`)	$\wedge, \vee, \neg$	

Expr($\underline{\text{Expr1}}$?$\underline{\text{Expr2}}$:$\underline{\text{Expr3}}$)	**if** *Expr*(Expr1) = **True**$\longrightarrow$
	Expr(Expr2)
	[] *Expr*(Expr3)
	fi	

Expr(($\underline{\text{Arg1}}$, $\underline{\text{Arg2}}$))	(*Expr*($\underline{\text{Arg1}}$), *Expr*($\underline{\text{Arg2}}$))	Translating argument tuples

Table 5.6: Expression Translation Rules

Rule №	Application	Result
*Meth*1	$\underline{\text{Obj.Meth}}$($\underline{\text{Args}}$);	Name($\underline{\text{Meth}}$)*Call* . Id($\underline{\text{Obj}}$) ! Expr($\underline{\text{Args}}$) $\longrightarrow$ Name($\underline{\text{Meth}}$)*Ret* . Id($\underline{\text{Obj}}$)$\longrightarrow$**Skip**
*Meth*2	$\underline{\text{Obj.Meth}}$($\underline{\text{Args}}$);	*binder_* Name($\underline{\text{Meth}}$)*Call* . Id($\underline{\text{Obj}}$) ! Expr($\underline{\text{Args}}$) $\longrightarrow$ *binder_* Name($\underline{\text{Meth}}$)*Ret* . Id($\underline{\text{Obj}}$)$\longrightarrow$**Skip**
*Meth*3	$\underline{\text{Obj.Meth}}$($\underline{\text{Args}}$);	Name($\underline{\text{Meth}}$) *Call* . Id($\underline{\text{Obj}}$) . Id(`this`) . TId($\underline{\text{Obj}}$) ! Expr($\underline{\text{Args}}$) $\longrightarrow$ Name($\underline{\text{Meth}}$)*Ret* . Id($\underline{\text{Obj}}$) . Id(`this`) . TId($\underline{\text{Obj}}$)$\longrightarrow$**Skip**

Table 5.7: Method Invocation Translation Rules

Attribute	Value
Name	“MainMission”
Parameters	[ ]
Variables	[name:“buffer”, type:“Buffer”, initialisation:“new Buffer()”]
Synchronised Methods	[ ]
Methods	[{name:“initialize”, parameters:null, returnType:void, returnValue:null, body:“$register!ProducerSID!MainMission \longrightarrow register!ConsumerSID!MainMission$”}, {name:“cleanUp”, parameters:null, returnType:boolean, returnValue:false, body:“”}, {name:“getBuffer”, parameters:null, returnType:“Buffer”, returnValue:[“buffer”], body:“”}]
Circus Class	{name:“MainMission”,parameters:[ ], variables:[name:“buffer”, type:“Buffer”, initialisation:“new Buffer()”], methods:[ ], synchronised methods:[ ]}
Schedulables	[“Producer”,“Consumer”]

Table 5.8: Environment of the `MainMission` Class

This is is specialised for the environments of missions.

Build Non-Paradigm Objects

The process of constructing a *ClassEnv* is the same for each non-paradigm object. The *ClassEnv*s for all of the non-paradigm objects in the program can be built at the beginning of the build phase. This is because non-paradigm objects are not involved in a program's hierarchy, so the order in which they are built is unimportant. This process translates the methods and variables of the object, using the rules described above to build its class environment. The list of *ClassEnv*s is stored in the program's *ProgramEnv*.

Build Safelet

The process of constructing a *SafeletEnv* for the safelet uses the rules described above to build its class environment and captures some additional information, which we describe in this section. To find the safelet, we consult the map constructed during step one of the analysis phase, which indicates which class implements the `Safelet` interface.

This process specifically translates the safelet's API methods `initializeApplication()` and `getSequencer()`. The *SafeletEnv* records the names of the mission sequencers that are returned by the `getSequencer()` method. These mission sequencers are the top-level mission sequencers. The *SafeletEnv* built by this process is stored in the *ProgramEnv*.

Build Mission Sequencer

The process to build a *MissionSequencerEnv* for a mission sequencers uses the rules described above to build its class environment and captures some extra information, which we describe in this section. This process is used for both top-level and schedulable mission sequencers. It captures a mission sequencer's `getNextMission()` method and application-defined variables and methods. The missions returned by the `getNextMission()` method are recorded in the *MissionSequencerEnv*. This list informs which missions are translated next.

The *MissionSequencerEnv* produced by this process is stored in the *ProgramEnv*. If the *MissionSequencerEnv* represents a top-level mission sequencer, then it is stored directly in the list in the *ProgramEnv*. If it is a schedulable mission sequencer, then it is stored in the list of schedules in the *ClusterEnv* that holds its controlling mission.

Build Mission

The process to build a *MissionEvn* for a mission uses the rules described above to build its class environment and captures some extra information, which we describe in this section. It captures a mission's `initialise()` and `cleanUp()` methods. This process also records the schedulables that are registered to this mission in its `initialise()` method, in the *MissionEnv*. This list informs the schedulables that are translated next. The environment built by this process is stored in its own *ClusterEnv*, where its schedulables will also be stored.

Build Schedulable

The process that builds a *ClassEnv* for a schedulable uses the rules described above to build its class environment. The schedulable is checked against the map constructed in step one of the analysis phase (Sect. 5.1.1 to find its type. If the schedulable is an event handler or managed thread, then a *ClassEnv* is built. If the schedulable is a mission sequencer, then the previously described process is used to build a *MissionSequencerEnv*. Any mission sequencers translated by this process are schedulable mission sequencers.

For an event handler, this process captures its `handleAsyncEvent()` method. For a managed thread, this process captures its `run()` method. A *ClassEnv* built by this process is stored in the *ClusterEnv* that holds its controlling mission. The environments of the missions and schedulables loaded by a schedulable mission sequencer are built before the process is complete.

5.1.3 Generate

The generate phase takes the environments produced by the build phase (Sect. 5.1.2) and outputs the *Circus* processes, *OhCircus* classes, and sets of channels that make up the application model. Generating the components of our model involves combining an environment with a template. This combination replaces tags in the template with the relevant information from the environment. There is a different template for each type of paradigm object, for non-paradigm objects, for *OhCircus* classes, and for channels.

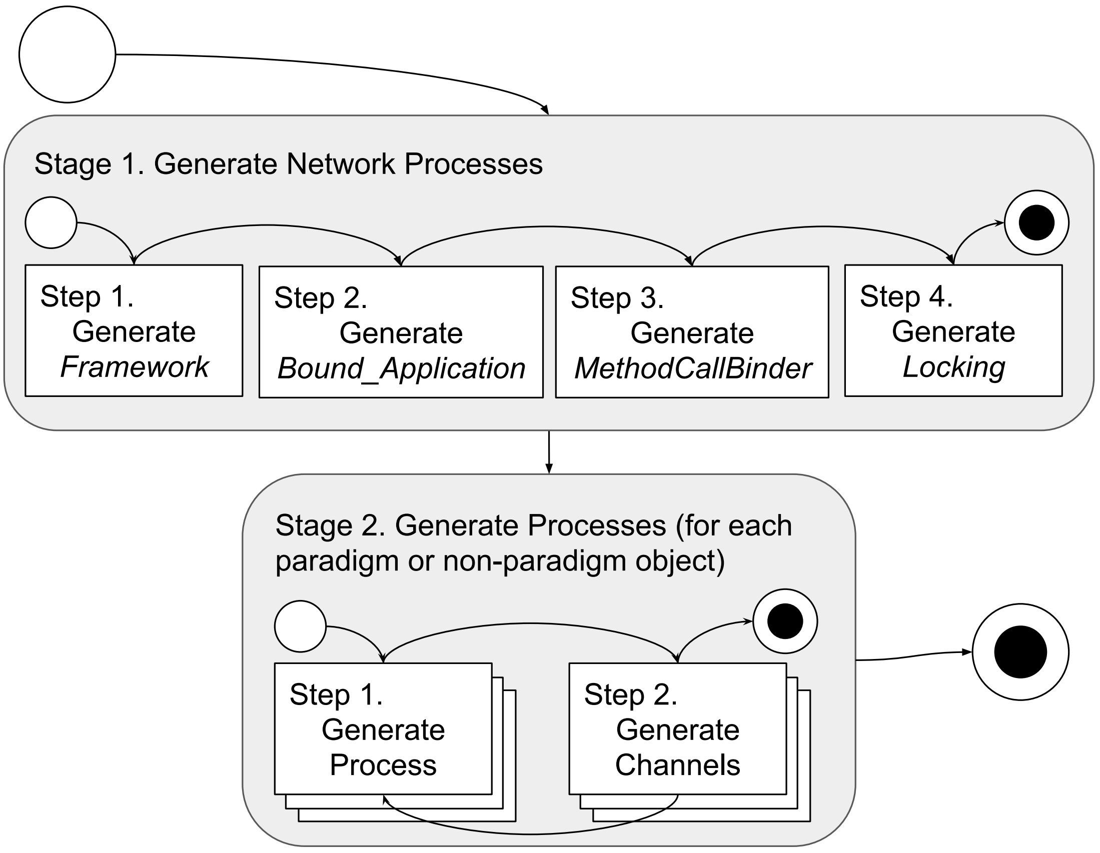

Stage 1. Generate Network Processes
Step 1. Generate *Framework*
Step 2. Generate *Bound_Application*
Step 3. Generate *MethodCallBinder*
Step 4. Generate *Locking*
Stage 2. Generate Processes (for each paradigm or non-paradigm object)
Step 1. Generate Process
Step 2. Generate Channels

Figure 5.5: Flow Diagram of the Generate Phase

Figure 5.5 the structure of this phase, which is split into two stages. Stage one produces the top-level network of processes that control our model. Stage two produces the *Circus* processes, *OhCircus* classes, and channels that model the objects in the SCJ program. Channels are generated for both the top-level processes generated in stage one and the processes representing the objects in the program, generated in stage two.

Stage One

Stage one occurs in four steps, illustrated in the top box of Fig. 5.5. It uses the information in the *NetworkEnv* environment, which records the *ProgramEnv*, *AppEnv*, and *LockingEnv* environments. These are used to construct the network of processes that make up the whole model: $\textbf{process}\, Program \mathrel{\widehat=} (Framework \llbracket appSync \rrbracket Bound\_Application) \llbracket lockSync \rrbracket Locking$. The *Framework* process controls all of the framework processes. The *Bound_Application* process contains the *Application* and *MethodCallBinder* processes; *Application* controls the application processes and *MethodCallBinder* provides actions that bind method calls to their locations. Finally, the *Locking* process controls the processes dealing with locking and suspension. The arrangement of the processes in the *Framework* process and the processes contained in the *Application* and *Locking* processes are driven by the information in the environments.

The channel set *appSync* contains the channels needed to allow communication between the *Framework* and *Application* processes. This set comprises all the channels that represent the framework handing control to the application. The *lockSync* channel set contains the channels that allow the *Framework* and *Application* processes to communicate with the *Locking* process. This set comprises the channels representing synchronisation, suspension,

$$\mathbf{process}\ \mathit{ControlTier} \mathrel{\widehat=} \left(\begin{array}{l} \mathit{SafeletFW} \\ \quad [\![\mathit{ControlTierSync}]\!] \\ \mathit{TopLevelMissionSequencerFW}(\underline{\mathsf{Name}}) \end{array}\right)$$

Figure 5.6: Control Tier Template

and thread interruption. Both of these channel sets are fixed and require no information from the SCJ program. Below, we describe the four steps in stage one of the generate phase, each generating a different component of the network.

Step One: Framework Process

The first step generates the *Framework* process, which controls the framework processes in the model. The framework processes are generic and reused for each translated program (as described in Chap 4). However, the application being translated dictates which framework processes are required and their structural arrangement.

Each process, except for the *SafeletFW* process, in instantiated with the identifier of its application counterpart, and any parameters required by the process. As previously mentioned, to make communication between the framework processes easier to specify, they are split into the *ControlTier* and a number of program tiers. The *ControlTier* contains the *SafeletFW* and *TopLevelMissionSequencerFW* processes in parallel. The template for the *ControlTier*, shown in Fig. 5.6, is combined with the *MissionSequencerEnv* representing the top-level mission sequencer to produce its model. In this simple example, that tag $\underline{\mathsf{Name}}$ is replaced by the name from the top-level mission sequencer's environment.

A program tier contain one or more clusters, with each cluster containing a *MissionFW* process and its schedulable framework processes. Fig: 5.7 shows the template for a program tier. Each *TierEnv* is combined with this template to produce the model of one program tier, by using the information in the environments recorded in the *TierEnv*. The $\underline{\mathsf{i}}$ tag is the index of the location of the tier being constructed in the list of *TierEnv*s, this produces a simple unique name for the process. For example, the first tier in the list is named 'Tier0', the second tier in the list is named 'Tier1', and so on.

For each *ClusterEnv* in the *TierEnv*, a cluster *Circus* process is generated, which consists of the name of the mission (which replaces the $\underline{\mathsf{MName}}$ tag) and the names and parameter lists of the schedulables registered to that mission. For each schedulable we instantiate the relevant framework process with its name and any API parameters. For example, a periodic event handler produces $\mathit{PeriodicEventHandlerFW}(\underline{\mathsf{PEHName}}\mathit{ID}, \underline{\mathsf{Params}})$, where the $\underline{\mathsf{PHEName}}$ tag is replaced by the name in the *ClassEnv* representing the handler, and the $\underline{\mathsf{Params}}$ tag is replaced by a list of the parameters to the periodic event handler constructor. For example, translating a periodic event handler with the name `Peh`, no start time offset, a period of 50ms, no deadline, and no deadline miss handler, produces the instantiation $\mathit{PeriodicEventHandlerFW}(\mathit{PehID}, \mathit{NULL}, \mathit{time}(50, 0), \mathit{NULL}, \mathit{nullSID})$. The NULL values indicate that no time has been specified, and the *nullSID* is a null identifier to indicate that

$$
\begin{array}{l}
\textbf{process}\ Tier\underline{\mathsf{i}} \mathrel{\widehat=} \\
\left(\begin{array}{l}
MissionFW(\underline{\mathsf{MName}}ID) \\
\quad [\![MissionSync]\!] \\
\left(\begin{array}{l}
PeriodicEventHandlerFW(\underline{\mathsf{PEHName}}ID, \underline{\mathsf{Params}}) \\
\quad [\![SchedulablesSync]\!] \\
\ldots \\
AperiodicEventHandlerFW(\underline{\mathsf{APEHName}}ID, \underline{\mathsf{Params}}) \\
\quad [\![SchedulablesSync]\!] \\
\ldots \\
OneShotEventHandlerFW(\underline{\mathsf{OSEHName}}ID, \underline{\mathsf{Params}}) \\
\quad [\![SchedulablesSync]\!] \\
\ldots \\
SchedulableMissionSequencerFW(\underline{\mathsf{SMSName}}ID) \\
\quad [\![SchedulablesSync]\!] \\
ManagedThreadFW(\underline{\mathsf{MTName}}ID) \\
\quad [\![SchedulablesSync]\!] \\
\ldots
\end{array}\right)
\end{array}\right) \\
\quad [\![ClusterSync]\!] \\
\ldots
\end{array}
$$

Figure 5.7: Program Tier Template

there is no deadline miss handler.

The *MissionSync*, *SchedulablesSync*, and *ClusterSync* channel sets require no information from the program, they are fixed. The *MissionSync* channel set allows the mission to communicate with its schedulables. It contains the channels for signalling the activation, registration, and termination of schedulables; the termination of the mission, top-level mission sequencer, and safelet; and, the channels modelling the schedulable's `cleanUp()` method. The *SchedulablesSync* channel set controls the communication between schedulables in the same cluster. It contains the channels that activate the schedulables and signal the termination of the safelet and top-level mission sequencer. Finally, the *ClusterSync* channel set controls the communication between clusters. It contains the channels that signal the termination of the safelet and top-level mission sequencer.

The *Framework* process uses the *TierSync* channel set to control the communication between the *ControlTier* and the program tiers, shown in Fig. 5.8. The process also uses tier-specific channel sets to control the communication between the program tiers (for example, *Tier*0*Sync* in Fig. 5.8), which contain the channels signalling termination of the safelet and top-level mission sequencer and some channels that are specific to each tier.

As previously mentioned, a program will always have a *Tier*0, containing the *MissionFw* processes that can be returned by the top-level mission sequencer. If a mission in *Tier*0 registers a schedulable mission sequencer, then *Tier*1 will contain the *MissionFW* processes that can be returned by that schedulable mission sequencer. A tier can contain *MissionFW*

$$\mathbf{process}\, \mathit{Framework} \mathrel{\widehat=} \left(\begin{array}{l} \mathit{ControlTier} \\ \quad \llbracket \mathit{TierSync} \rrbracket \\ \left(\begin{array}{l} \mathit{Tier}0 \\ \quad \llbracket \mathit{Tier}0\mathit{Sync} \rrbracket \\ \mathit{Tier}1 \end{array} \right) \end{array} \right)$$

Figure 5.8: Example of the *Framework* Process

$$\mathbf{process}\, \mathit{Application} \mathrel{\widehat=} \left(\begin{array}{l} \underline{\mathsf{Name}}\mathit{App}(\underline{\mathsf{Params}}) \\ ||| \\ \ldots \end{array} \right)$$

Figure 5.9: *Application* Process Template

processes that are not loaded by the same mission sequencer. For example, if a mission in *Tier*0 registers two schedulable mission sequencers, then the missions of both of those mission sequencers would be in *Tier*1. As an example, Fig. 5.8 shows the *Framework* process for a program that has at lest one schedulable mission sequencer and hence has two tiers.

To generate the synchronisation set that controls communication between tiers, we use the identifier from each *MissionEnv* in the tier below to restrict the channels controlling the start and termination of missions with the identifier of the missions in this tier. If we are generating a channel set for a program tier that is not *Tier*0, then we also use the identifiers from any *MissionSequencerEnv* environments in the tier below to restrict the channel requesting mission termination with the identifiers of the missions and mission sequencers in this tier. For example, generating the *TierSync* set in Fig. 5.8, where *Tier*0 contains the mission `MainMission` produces the set $\{\!|\ \mathit{done\_toplevel\_sequencer},\ \mathit{done\_safeletFW},\ \mathit{start\_mission.MainMissionID},\ \mathit{done\_mission.MainMissionID},\ \mathit{initializeRet.MainMissionID}\ |\!\}$.

Step Two: Application Process

The second step generates the *Application* process, which is one half of the *Bound_Application* process. The *Application* process is simpler to generate than the *Framework* process, because it is an interleaving of all the application processes that represent the program being modelled. The template for the *Application* process is shown in Fig. 5.9, and it is combined with the information in the *AppEnv* environment, produced by the build phase. The *AppEnv* lists pairs of process names and parameters. We combine each pair with the template $\underline{\mathsf{Name}}\mathit{App}(\underline{\mathsf{Params}})$, replacing the $\underline{\mathsf{Name}}$ tag with the process name and the $\underline{\mathsf{Params}}$ tag with the list of application-defined parameters. For example, generating an application process representing a managed thread `Thread`, which takes the identifier of the mission `MainMission` as a parameter, produces the process $\mathit{ThreadApp}(\mathit{MainMissionID})$.

$\mathbf{process}\ \mathit{MethodCallBinder} \mathrel{\widehat=} \mathbf{begin}$

$$\underline{\mathsf{Name}}\_\mathit{MethodBinder} \mathrel{\widehat=}$$
$$\left(\begin{array}{l}
\mathit{binder}\_\underline{\mathsf{Name}}\mathit{Call} \\
\quad ?\,\mathit{loc} : (\mathit{loc} \in \underline{\mathsf{Name}}\mathit{Locs}) \\
\quad ?\,\mathit{caller} : (\mathit{caller} \in \underline{\mathsf{Name}}\mathit{Callers}) \\
\quad ?\,\underline{\mathsf{Params}} \longrightarrow \\
\underline{\mathsf{Name}}\mathit{Call} \,.\, \mathit{loc} \,.\, \mathit{caller}\,!\,\underline{\mathsf{Params}} \longrightarrow \\
\underline{\mathsf{Name}}\mathit{Ret} \,.\, \mathit{loc} \,.\, \mathit{caller}\,?\,\mathit{ret} \longrightarrow \\
\mathit{binder}\_\underline{\mathsf{Name}}\mathit{Ret} \,.\, \mathit{loc} \,.\, \mathit{caller}\,!\,\mathit{ret} \longrightarrow \\
\underline{\mathsf{Name}}\_\mathit{MethodBinder}
\end{array}\right)$$

$\ldots$

$$\mathit{BinderActions} \mathrel{\widehat=}$$
$$\left(\begin{array}{l}
\underline{\mathsf{Name}}\_\mathit{MethodBinder} \\
\interleave \\
\ldots
\end{array}\right)$$

$\bullet\ \mathit{BinderActions} \mathbin{\triangle} (\mathit{done\_toplevel\_sequencer} \longrightarrow \mathbf{Skip})$

$\mathbf{end}$

Figure 5.10: Template for the *MethodCallBinder* Process

Step Three: MethodCallBinder Process

The third step generates the *MethodCallBinder* (*MCB*) process, which is the second half of the *Bound_Application*, and its associated channels and sets. The *MCB* contains an action for each of the application-defined methods in the program, as described in Sect. 4.4. Figure 5.10 shows the template for the *MethodCallBinder* process. Each action in the process is generated using the information in one *MethodEnv*. The $\underline{\mathsf{Name}}$ and the $\underline{\mathsf{Params}}$ tags in the template are replaced by the name and parameter list from the *MethodEnv*, respectively. If the *MethodEnv* represents a synchronised method, then the channels in the action generated also take a thread identifier as a parameter, to ensure that the communication refers to the correct *ThreadFW* process (as described in Sect. 4.3). This is achieved using the rules in Table 5.7.

This step also produces the call and return channels and the '*Locs*' and '*Callers*' sets for each of the *MCB* process's actions. The purpose of the '*Locs*' and '*Callers*' sets is described in full in Sect. 4.4 and recapped below. As shown in Fig. 5.10, the call and return channels are generated using the name of the bound method. They both accept a *loc* and a *caller* parameter. The call channel accepts the same parameters as the method's call channel, and the return channel accepts a return parameter of the same type as the method's return channel. The *Locs* set contains the identifiers of the processes that contain the method, and the *Callers* set contains the identifiers of the processes that call the method. These sets are prefixed by the method name, to produce a unique name. The '*binder*' call event only accepts communications when the *loc* and *caller* parameters are in its associated '*Locs*' and '*Callers*' sets, respectively.

The *Locs* set for a particular method contains the names of all of the classes in the Method Class Map (constructed during step two of the analysis phase) that map to a list that contains the method's name. As previously mentioned, to deal with name clashes, two methods are considered to be the same if their names are the same and they are declared in the same class or classes related by inheritance. In such cases, those methods share call and return channels and produce one '*_MethodBinder*' action. If two methods share only the same name, but are not declared in classes related by inheritance, then the method name is prefixed by the class name to produce unique action and channel names.

The *Callers* set for a particular method is the list of class names mapped to by that method in the Method Callers Map (constructed during step two of the analysis phase). During the building of the Method Callers Map, the same name clash precautions have been taken as for building the *Locs* set, above.

Step Four: Locking Process

The fourth step generates the *Locking* process. It contains the *Objects* and *Threads* process in parallel, synchronising on *ThreadSync*, which contains the channels that allow *ObjectFW* processes to alter or query a *ThreadFW*'s priority. This allows our model to capture Priority Ceiling Emulation, as used by SCJ programs. However, we only check for priority exceptions during suspension.

The *LockingEnv* environment is combined with the template for the *Threads*, *Objects*, and *Locking* processes, shown in Fig. 5.11, to produce the processes. The $\underline{\mathsf{TName}}$ and $\underline{\mathsf{TPriority}}$ tags are replaced with the names and priories of each of the *ThreadFW* processes. The $\underline{\mathsf{OName}}$ tags are replaced with the names of each of the *ObjectFW* processes.

Stage Two

In stage two, for each environment representing a paradigm or non-paradigm object, we take two steps. Because the environments already contain the information required to produce the model, this phase can approach the environments in any order. The first step generates a process, and possibly an *OhCircus* class, from an environment. The second step generates the channels required by the processes generated by the first step.

Step One: Processes

The first step takes each environment, identifies the required output template, and combines them to produce a process (and possibly an *OhCircus* class). The required output template can be determined by checking the type of the environment in Table 5.1.

Figure 5.12 shows the generic template for all application processes; it shows the structure of the generic application process templates and is used to as the template non-paradigm

$$\mathbf{process}\ Threads \mathrel{\widehat=} \left(\begin{array}{l} ThreadFW(\underline{\mathsf{TName}}ID, \underline{\mathsf{TPriority}}) \\ ||| \\ \dots \end{array}\right)$$

$$\mathbf{process}\ Objects \mathrel{\widehat=} \left(\begin{array}{l} ObjectFW(\underline{\mathsf{OName}}ID) \\ ||| \\ \dots \end{array}\right)$$

$$\mathbf{process}\ Locking \mathrel{\widehat=} Threads \llbracket ThreadSync \rrbracket Objects$$

Figure 5.11: *Threads*, *Objects*, and *Locking* Process Teamples

objects. The $\underline{\mathsf{processName}}$ tag is replaced by the name in the *ClassEnv* and the $\underline{\mathsf{params}}$ tag is replaced by the list of parameters. For each of the *VarEnv*s in the *ClassEnv*'s variable list we produce a variable in the process's state and an entry in the *init* schema (if the variable is initialised in the program). To do this, we replace the $\underline{\mathsf{varName}}$ tags with the variable's name, the $\underline{\mathsf{varType}}$ tags with the variable type, and $\underline{\mathsf{varInit}}$ with the variable's initialisation. Finally, the $\underline{\mathsf{processType}}$ is replaced by the type of process (the type of paradigm object it models except for non-paradigm objects, which use 'safelet' so that they are terminated along with the safelet process) and the $\underline{\mathsf{processID}}$ tag is replaced by the identifier of the process.

Similarly, for a class's methods, we produce one action per *MethodEnv* in the *ClassEnv*'s *Meth* or *SyncMeth* list – which contain the environments of the class's method and synchronised methods, respectively. To do this, we replace the $\underline{\mathsf{methName}}$ tags with the name of the method, the $\underline{\mathsf{returnType}}$ tag with the method's return type, and the $\underline{\mathsf{methBody}}$ tags with the body of the method. Note that if the method has no return type, then the return variable ($\mathbf{var}\ \underline{\mathsf{returnType}} : ret$) is omitted from the action definition.

Each type of object requires its own template. These specific templates are based on the generic template in Fig. 5.12, extended with local actions for its specific API-defined methods. We describe these specific templates below.

To generate the safelet application process, we combine a *SafeletEnv* with the template in Fig. 5.13. In addition to the tags present in the generic template, when using this template we replace the $\underline{\mathsf{initBody}}$ tag with the body of the safelet's `initializeApplication()` method and the $\underline{\mathsf{SchedulableID}}$ tag with the identifier of the top-level mission sequencer that is returned by the safelet's `getSequencer()` method. The information required by both of these tags is found in the *MethodEnv* representing the relevant method in the *SafeletEnv*.

To generate a mission sequencer application process we combine a *MissionSequenecerEnv* with the template in Fig. 5.14. This template adds an action to model the `getNextMission()` method, but contains no tags that are not present in the generic template. The state of a mission sequencer application process contains an *OhCircus* class, which defines the variables

$\mathbf{process}\ \underline{\mathsf{processName}}App \mathrel{\widehat=} \underline{\mathsf{params}}\ \mathbf{begin}$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{State}} \\ \hline
\underline{\mathsf{varName}} : \underline{\mathsf{varType}} \\
\ldots \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Init}} \\ \hline
\textit{State}\,' \\ \hline
\underline{\mathsf{varName}}' = \underline{\mathsf{varInit}} \\
\ldots \\ \hline
\end{array}$$

$\underline{\mathsf{methName}} \mathrel{\widehat=}$
$\mathbf{var}\ \underline{\mathsf{returnType}} : ret \bullet \left(\underline{\mathsf{methBody}}\right)$

$\ldots$

$$Methods \mathrel{\widehat=} \left(\begin{array}{l} \underline{\mathsf{methName}} \\ \Box \\ \ldots \end{array}\right) ;\ Methods$$

$\bullet\ (Init\ ;\ Methods) \mathbin{\triangle} (end\_\underline{\mathsf{processType}}\_app\,.\,\underline{\mathsf{processID}} \longrightarrow \mathbf{Skip})$

Figure 5.12: Generic Application Process Template

and non-reactive behaviour of the mission sequencer object. An example of this can be seen in the *GetNextMission* action in Fig. 5.14, which calls the *getNextMission*() *OhCircus* method in the class referenced by the variable *this*. The method contains a translation of the mission sequencer object's `getNextMission()` method, because it does not interact with other processes.

To generate a mission application process we combine a *MissionEnv* with the template shown in Fig. 5.15. In addition to the generic template, the mission template adds two extra actions. The *InitializePhase* action represents the `initialize()` method. In this action, for each schedulable in the *Schedulables* list in the source *MissionEnv*, we generate a *register* event, where the $\underline{\mathsf{SID}}$ tag is replaced with the identifier of the schedulable.

The *CleanupPhase* action represents the `cleanUp()` method, which returns a boolean, so the action defines a boolean *ret* variable that is output on the return channel. The $\underline{\mathsf{cleanupBody}}$ tag is replaced with the body of the `cleanUp()` method. It is expected that the *ret* variable will be set in the method body, but if it is not then $ret := \mathbf{True};$ is added before the return channel.

As an example of the output of the generate phase, we return to the `MainMission` from the buffer application, shown in Fig. 5.2. To produce the model for the `MainMisison` class, the generate phase combines the template in Fig. 5.15 with the information from the environment

process $\underline{\mathsf{processName}}App \mathrel{\widehat=} \underline{\mathsf{params}}$ **begin**

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{State}} \\ \hline
\underline{\mathsf{varName}} : \underline{\mathsf{varType}} \\
\ldots \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Init}} \\ \hline
\mathit{State}' \\ \hline
\underline{\mathsf{varName}}' = \underline{\mathsf{varInit}} \\
\ldots \\ \hline
\end{array}$$

$$\begin{array}{l}
\mathit{InitializeApplication} \mathrel{\widehat=} \\
\left(\begin{array}{l}
\mathit{initializeApplicationCall} \longrightarrow \\
\underline{\mathit{initBody}} \\
\mathit{initializeApplicationRet} \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}$$

$$\begin{array}{l}
\mathit{GetSequencer} \mathrel{\widehat=} \\
\left(\begin{array}{l}
\mathit{getSequencerCall} \longrightarrow \\
\mathit{getSequencerRet}\,!\,\underline{\mathsf{SchedulableID}} \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}$$

$$\begin{array}{l}
\underline{\mathsf{methName}} \mathrel{\widehat=} \\
\mathbf{var}\ \underline{\mathsf{returnType}} : \mathit{ret} \bullet \left(\underline{\mathsf{methBody}}\right)
\end{array}$$

$\ldots$

$$\begin{array}{l}
\mathit{Methods} \mathrel{\widehat=} \\
\left(\begin{array}{l}
\mathit{GetSequencer} \\
\Box \\
\mathit{InitializeApplication} \\
\Box \\
\underline{\mathsf{methName}} \\
\Box \\
\ldots
\end{array}\right) ;\ \mathit{Methods}
\end{array}$$

$\bullet\ (\mathit{Init}\ ;\ \mathit{Methods}) \mathbin{\triangle} (\mathit{end\_safelet\_app} \longrightarrow \mathbf{Skip})$

Figure 5.13: Template for Safelet Application Processes

$\mathbf{process}\ \underline{\mathsf{processName}}App \mathrel{\widehat=} \underline{\mathsf{params}}\ \mathbf{begin}$

State

$this : \mathrm{ref}\ \underline{\mathsf{processName}}Class$

$\underline{\mathsf{varName}} : \underline{\mathsf{varType}}$

$\ldots$

Init

$State\,'$

$this = \mathbf{new}\ \underline{\mathsf{processName}}Class()$

$\underline{\mathsf{varName}}' = \underline{\mathsf{varInit}}$

$\ldots$

$$GetNextMission \mathrel{\widehat=} \mathbf{var}\ ret : MissionID \bullet \left(\begin{array}{l} getNextMissionCall\,.\,\underline{\mathsf{processName}}ID \longrightarrow \\ ret := this\,.\,getNextMission(); \\ getNextMissionRet\,.\,\underline{\mathsf{processName}}ID\,!\,ret \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$\underline{\mathsf{methName}} \mathrel{\widehat=} \mathbf{var}\ \underline{\mathsf{returnType}} : ret \bullet \left(\underline{\mathsf{methBody}}\right)$$

$\ldots$

$$Methods \mathrel{\widehat=} \left(\begin{array}{l} GetNextMission \\ \Box \\ \underline{\mathsf{methName}} \\ \Box \\ \ldots \end{array}\right) ;\ Methods$$

$$\bullet\ (Init\ ;\ Methods) \mathbin{\triangle} (end\_sequencer\_app\,.\,\underline{\mathsf{ProcessName}}ID \longrightarrow \mathbf{Skip})$$

Figure 5.14: Template for Mission Sequencer Application Process

$\mathbf{process}\ \underline{\mathsf{processName}}App \mathrel{\widehat=} \underline{\mathsf{params}}\ \mathbf{begin}$

State

$\underline{\mathsf{varName}} : \underline{\mathsf{varType}}$

$\ldots$

Init

$State~'$

$\underline{\mathsf{varName}}' = \underline{\mathsf{varInit}}$

$\ldots$

$$InitializePhase \mathrel{\widehat=} \left(\begin{array}{l} initializeCall \,.\, \underline{\mathsf{ProcessName}}ID \longrightarrow \\ register \,!\, \underline{\mathsf{SchedulableID}} \,!\, \underline{\mathsf{ProcessName}}ID \longrightarrow \\ \ldots \\ initializeRet \,.\, \underline{\mathsf{ProcessName}}ID \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$CleanupPhase \mathrel{\widehat=} \mathbf{var}\ \mathbb{B} : ret \bullet \left(\begin{array}{l} cleanupMissionCall \,.\, \underline{\mathsf{ProcessName}}ID \longrightarrow \\ \underline{cleanupBody}\, cleanupMissionRet \,.\, \underline{\mathsf{ProcessName}}ID \,!\, ret \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$\underline{\mathsf{methName}} \mathrel{\widehat=} \mathbf{var}\ \underline{\mathsf{returnType}} : ret \bullet \left(\underline{\mathsf{methBody}}\right)$$

$\ldots$

$$Methods \mathrel{\widehat=} \left(\begin{array}{l} InitializePhase \\ \Box \\ CleanupPhase \\ \Box \\ \underline{\mathsf{methName}} \\ \Box \\ \ldots \end{array}\right) ;\ Methods$$

$\bullet\ (Init\ ;\ Methods) \bigtriangleup (end\_mission\_app \,.\, \underline{\mathsf{ProcessName}}ID \longrightarrow \mathbf{Skip})$

Figure 5.15: Template for Mission Application Processes

described in Table. 5.8. The result of this combination generates the model *MainMissionApp* shown in Fig. 5.16.

To generate a process for an event handler we combine a *ClassEnv* representing the event handler with the template shown in Fig. 5.17. This template is used to translate all three types of event handler. This is possible because the specifics of their release patterns are captured by the framework model. In addition to the generic template, the event handler template adds an action to model the handler's `handleAsyncEvent()` method, in which we replace the HandleAsyncBody tag with the body of the method. The method body is taken from the *MethodEnv* that represents the `handleAsyncEvent()` method in the *ClassEnv* for this handler. Finally, the HandlerType tag is replaced with name of the type of handler (*periodic*, *aperiodic*, or *oneshot*) to complete the name of the channel that terminates it.

To generate a process for a managed thread we combine a *ClassEnv* that represents a managed thread object with the template shown in Fig 5.18. In addition to the generic template, the managed thread template adds an action to model the `run()` method, in which we replace the RunBody tag with the body of the action (found in the *MethodEnv* representing that method in the *ClassEnv*).

Step Two: Channels

The second step generates the channels required by the processes produced in the first step. These channels represent the call to and return from the actions in each application process. Figure 5.19 shows the template for generating the pair of channels for non-synchronised methods and 5.20 shows the templates for generating the pair of channels for synchronised methods. We combine one of these templates with the information in a *MethodEnv*, replacing each tag with the relevant information.

The ChanName tag is replaced with the name of the channel (which is the name of the method is representing). The LocType tag is replaced with the type of the identifier used by the location of the method and the CallerType tag with the type of the identifier used by the callers of the method. If the method is only called inside the class it's defined in, then the '×CallerType' is omitted. The identifier type will be either *appID*, for non-paradigm objects; *MissionID*, for missions; or *SchedulableID*, for schedulables. The *CallerType* and *LocType* are used to handle inheritance and polymorphism, as described in Sect. 4.4. For the call channel, we replace the Params tag with any parameters of the method it represents. Finally, for the return channel, we replace the ReturnType tag with the return type of the method it is representing.

5.2 Core Formalisation

In this section we present a formalisation in Z of the core elements of the translation strategy described in Sect. 5.1. Z provides a useful logical framework with which to formalise our translation. At the top-level, we define a function, *TransSCJProg* that accepts an SCJ Level 2 program and outputs its *Circus* model. To give us the facility to describe SCJ and *Circus* programs, we encode a BNF description of both SCJ and *Circus* into Z. This encoding

process *MainMissionApp* $\widehat{=}$ **begin**

State
$this : \mathbf{ref}\ MainMissionClass$

Init
$State'$
$this' = \mathbf{new}\ MainMissionClass()$

$$InitializePhase \mathrel{\widehat{=}} \left(\begin{array}{l} initializeCall\,.\,MainMissionMID \longrightarrow \\ register\,!\,ProducerSID\,!\,MainMissionMID \longrightarrow \\ register\,!\,ConsumerSID\,!\,MainMissionMID \longrightarrow \\ initializeRet\,.\,MainMissionMID \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$CleanupPhase \mathrel{\widehat{=}} \\ \mathbf{var}\ ret : \mathbb{B} \bullet \left(\begin{array}{l} cleanUpCall\,.\,MainMissionMID \longrightarrow \\ ret := this\,.\,cleanUp(); \\ cleanUpRet\,.\,MainMissionMID\,!\,ret \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$getBufferMeth \mathrel{\widehat{=}} \mathbf{var}\ ret : Buffer \bullet \left(\begin{array}{l} getBufferCall\,.\,MainMissionMID \longrightarrow \\ ret := this\,.\,getBuffer(); \\ getBufferRet\,.\,MainMissionMID\,!\,ret \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$Methods \mathrel{\widehat{=}} \left(\begin{array}{l} InitializePhase \\ \Box \\ CleanupPhase \\ \Box \\ getBufferMeth \end{array}\right) ;\ Methods$$

$$\bullet\ (Init\ ;\ Methods) \mathbin{\triangle} (end\_mission\_app\,.\,MainMissionMID \longrightarrow \mathbf{Skip})$$

Figure 5.16: Example of *MainMissionApp* Processes Generated by the Translation Strategy

$\mathbf{process}\ \underline{\mathsf{processName}}App \mathrel{\widehat=} \underline{\mathsf{params}}\ \mathbf{begin}$

State

$\underline{\mathsf{varName}} : \underline{\mathsf{varType}}$

$\ldots$

Init

$State\,'$

$\underline{\mathsf{varName}}' = \underline{\mathsf{varInit}}$

$\ldots$

$$handleAsyncEvent \mathrel{\widehat=} \left(\begin{array}{l} handleAsyncEventCall\,.\,\underline{\mathsf{processName}}ID \longrightarrow \\ \underline{\mathsf{HandleAsyncBody}}; \\ handleAsyncEventRet\,.\,\underline{\mathsf{processName}}ID \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$\underline{\mathsf{methName}} \mathrel{\widehat=} \mathbf{var}\ \underline{\mathsf{returnType}} : ret \bullet \left(\underline{\mathsf{methBody}}\right)$$

$\ldots$

$$Methods \mathrel{\widehat=} \left(\begin{array}{l} handleAsyncEvent \\ \Box \\ \underline{\mathsf{methName}} \\ \Box \\ \ldots \end{array}\right) ;\ Methods$$

$$\bullet\ (Init\ ;\ Methods) \mathbin{\triangle} (end\_\underline{\mathsf{HandlerType}}\_app\,.\,\underline{\mathsf{processName}}ID \longrightarrow \mathbf{Skip})$$

Figure 5.17: Template for an Event Handler Application Process

$\mathbf{process}\ \underline{\mathsf{processName}}App \mathrel{\widehat=} \underline{\mathsf{params}}\ \mathbf{begin}$

$$\begin{array}{|l}
\multicolumn{1}{l}{State} \\ \hline
\underline{\mathsf{varName}} : \underline{\mathsf{varType}} \\
\ldots \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{S\_Init} \\ \hline
State~' \\ \hline
\underline{\mathsf{varName}}' = \underline{\mathsf{varInit}} \\
\ldots \\ \hline
\end{array}$$

$$\begin{array}{l}
Run \mathrel{\widehat=} \\
\left(\begin{array}{l}
runCall\,.\,\underline{\mathsf{processName}}ID \longrightarrow \\
\underline{\mathsf{RunBody}}; \\
runRet\,.\,\underline{\mathsf{processName}}ID \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}$$

$$\begin{array}{l}
\underline{\mathsf{methName}} \mathrel{\widehat=} \\
\mathbf{var}\ \underline{\mathsf{returnType}} : ret \bullet \left(\underline{\mathsf{methBody}}\right)
\end{array}$$

$\ldots$

$$\begin{array}{l}
Methods \mathrel{\widehat=} \\
\left(\begin{array}{l}
Run \\
\Box \\
\ldots
\end{array}\right) ;\ Methods
\end{array}$$

$\bullet\ (Init\ ;\ \ Methods) \mathbin{\triangle} (end\_managedThread\_app\,.\,\underline{\mathsf{processName}}ID \longrightarrow \mathbf{Skip})$

Figure 5.18: Template for a Managed Thread Application Process

$\mathbf{channel}\ \underline{\mathsf{ChanName}}Call : \underline{\mathsf{LocType}} \times \underline{\mathsf{CallerType}}\underline{\mathsf{Params}}$
$\mathbf{channel}\ \underline{\mathsf{ChanName}}Ret : \underline{\mathsf{LocType}} \times \underline{\mathsf{CallerType}} \times \underline{\mathsf{ReturnType}}$

Figure 5.19: Template for Non-Synchronised Method's Call and Return Channels

$$\textbf{channel}\ \underline{\mathsf{ChanName}}\mathit{Call} : \underline{\mathsf{LocType}} \times \underline{\mathsf{CallerType}} \times \mathit{ThreadID}\underline{\mathsf{Params}}$$
$$\textbf{channel}\ \underline{\mathsf{ChanName}}\mathit{Ret} : \underline{\mathsf{LocType}} \times \underline{\mathsf{CallerType}} \times \mathit{ThreadID} \times \underline{\mathsf{ReturnType}}$$

Figure 5.20: Template for Synchronised Method's Call and Return Channels

uses the BNF presented in Fig. 2.5 as its base, and is presented in Appendix. D.

Figure 5.21 shows the SCJ BNF encoded in Z. As previously mentioned, we consider that the paradigm of SCJ is separate from its implementation in Java. Therefore, the SCJ BNF encoding shows only the syntactic categories specific to SCJ programs. The definition of Java syntactic categories remains the same as in the Java Language Specification (JLS) [34].

In the Z encoding of the SCJ BNF, the *SCJProgram* type models an SCJ program as a *Safelet*, a *TopLevelMissionSequencer*, and a sequence of *Tier* values. The *Safelet* type defines the safelet as an *Identifier* and a *SafeletClassBody*. An *Identifier* has the same definition as in the JLS [34]. The *SafeletClassBody* defines the body of a safelet class as an *InitializeApplication*, *GetSequencer*, and *ImmortalMemorySize*, which represent the API methods defined by the `Safelet` interface, and a sequence of *ClassBodyDeclaration* values. Each of these is defined as a *MethodBody*, which has the same definition as in the JLS [34]. The sequence of *ClassBodyDeclaration* values represents the application-specific contents of the class, where a *ClassBodyDeclaration* has the same definition as in the JLS [34]. Each of the paradigm classes is encoded using this pattern, where each is defined as an identifier and the body of the relevant class.

TopLevelMissionSequencer defines the top-level mission sequencer as either *NoSequencer* (because a safelet may return `null` from its `getSequencer()` method) or a mission sequencer value defined by the constructor $tlms\langle\langle MissionSequencer\rangle\rangle$. For simplicity, this assumes that there is only one top-level mission sequencer, unlike the environments presented in Fig. 5.4. A *MissionSequencer* is defined with the same pattern as the *Safelet* category: it is an *Identifier* and a *MissionSequencerClassBody*, which contains values to represent the `getNextMission()` method and a sequence of *ClassBodyDeclaration* values.

As described at the start of Sect. 5.1.2, SCJ programs are organised into a *control tier* and several *program tiers*. Each program tier comprises several *clusters*, which each contain a mission and its registered schedulables. In the SCJ BNF encoding the *Safelet* and *TopLevelMissionSequencer* objects compose the control tier and the *Tier* object represents a program tier. A *Tier* is defined as a sequence of *Cluster* values, which comprises a *Mission* and a sequence of *SchedulableObject*s.

A *Mission* value defines a mission, using the same pattern as the previously described paradigm objects. A *SchedulableObject* constructs one of the schedulable object types: either an event handler ($handler\langle\langle EventHandler\rangle\rangle$), a managed thread ($mt\langle\langle ManagedThread\rangle\rangle$), or a mission sequencer ($nms\langle\langle NestedMissionSequencer\rangle\rangle$). An *EventHandler* is defined as one of the three types representing an event handler (periodic, aperiodic, or one-shot). Each of the handler categories, the *ManagedThread*, and the *NestedMissionSequencer* are defined with the same patternas the other paradigm objects we describe above. The *MethodDeclaration*, *Long*, and *ClassDeclaration* types have the same definition as in the JLS [34].

$$
\begin{array}{lll}
\mathit{SCJProgram} & == & \mathit{Safelet} \times \mathit{TopLevelMissionSequencer} \times \mathrm{seq}\ \mathit{Tier} \\
\mathit{Safelet} & == & \mathit{Identifier} \times \mathit{SafeletClassBody} \\
\mathit{SafeletClassBody} & == & \mathit{InitializeApplication} \times \mathit{GetSequencer} \times \\
 & & \mathit{ImmortalMemorySize} \times \mathrm{seq}\ \mathit{ClassBodyDeclaration} \\
\mathit{GetSequencer} & == & \mathit{MethodBody} \\
\mathit{InitializeApplication} & == & \mathit{MethodBody} \\
\mathit{ImmortalMemorySize} & == & \mathit{MethodDeclaration} \\
\mathit{TopLevelMissionSequencer} & ::= & \mathit{NoSequencer} \mid \mathit{tlms}\langle\!\langle \mathit{MissionSequencer} \rangle\!\rangle \\
\mathit{Tier} & == & \mathrm{seq}\ \mathit{Cluster} \\
\mathit{Cluster} & == & \mathit{Mission} \times \mathrm{seq}\ \mathit{SchedulableObject} \\
\mathit{Mission} & == & \mathit{Identifier} \times \mathit{MissionClassBody} \\
\mathit{MissionClassBody} & == & \mathit{Initialize} \times \mathit{Cleanup} \times \mathrm{seq}\ \mathit{ClassBodyDeclaration} \\
\mathit{Initialize} & == & \mathit{MethodBody} \\
\mathit{Cleanup} & == & \mathit{MethodBody} \\
\mathit{SchedulableObject} & ::= & \mathit{handler}\langle\!\langle \mathit{EventHandler} \rangle\!\rangle \mid \mathit{mt}\langle\!\langle \mathit{ManagedThread} \rangle\!\rangle \mid \\
 & & \mathit{nms}\langle\!\langle \mathit{NestedMissionSequencer} \rangle\!\rangle \\
\mathit{NestedMissionSequencer} & == & \mathit{MissionSequencer} \\
\mathit{MissionSequencer} & == & \mathit{Identifier} \times \mathit{MissionSequencerClassBody} \\
\mathit{MissionSequencerClassBody} & == & \mathit{GetNextMission} \times \mathrm{seq}\ \mathit{ClassBodyDeclaration} \\
\mathit{GetNextMission} & == & \mathit{MethodBody} \\
\mathit{EventHandler} & ::= & \mathit{pehDecl}\langle\!\langle \mathit{PeriodicEventHandler} \rangle\!\rangle \mid \\
 & & \mathit{apehDecl}\langle\!\langle \mathit{AperiodicEventHandler} \rangle\!\rangle \mid \\
 & & \mathit{osehDecl}\langle\!\langle \mathit{OneShotEventHandler} \rangle\!\rangle \\
\mathit{PeriodicEventHandler} & == & \mathit{Identifier} \times \mathit{EventHandlerClassBody} \\
\mathit{AperiodicEventHandler} & ::= & \mathit{apehType}\langle\!\langle \mathit{Identifier} \times \mathit{EventHandlerClassBody} \rangle\!\rangle \mid \\
 & & \mathit{aplehType}\langle\!\langle \mathit{Identifier} \times \mathit{LongEventHandlerClassBody} \rangle\!\rangle \\
\mathit{LongEventHandlerClassBody} & == & \mathit{HandleAsyncLongEvent} \times \mathrm{seq}\ \mathit{ClassBodyDeclaration} \\
\mathit{OneShotEventHandler} & == & \mathit{Identifier} \times \mathit{EventHandlerClassBody} \\
\mathit{EventHandlerClassBody} & == & \mathit{HandleAsyncEvent} \times \mathrm{seq}\ \mathit{ClassBodyDeclaration} \\
\mathit{HandleAsyncLongEvent} & == & \mathit{Long} \times \mathit{MethodBody} \\
\mathit{HandleAsyncEvent} & == & \mathit{MethodBody} \\
\mathit{ManagedThread} & == & \mathit{Identifier} \times \mathit{ManagedThreadClassBody} \\
\mathit{ManagedThreadClassBody} & == & \mathit{Run} \times \mathrm{seq}\ \mathit{ClassBodyDeclaration} \\
\mathit{Run} & == & \mathit{MethodBody}
\end{array}
$$

$$[\mathit{MethodBody}, \mathit{ClassBodyDeclaration}, \mathit{Identifier}, \mathit{MethodDeclaration}, \mathit{Long}, \mathit{ClassDeclaration}]$$

Figure 5.21: SCJ BNF Translated to Z

Model Component	Build Function	Generate Function
Framework Processes	*BuildFWEnv*	*GenerateFWProcs*
Application Process	*BuildAppProcEnv*	*GenerateAppProc*
MethodCallBinder Process	*BuildMCBEnv*	*GenerateMCBProc*
Locking Processes	*BuildLockEnv*	*GenerateLockProc*

Table 5.9: Functions used by the *TransSCJProg* Function

The formalisation captures the same three phases as the translation strategy in Sect. 5.1. The functions defining the analysis phase extract information from the program. The functions defining the build phase construct an environment for each of the objects in the program. Finally, the functions defining the generate phase construct the *Circus* models from the information in the environments. The build and generate functions use various other auxiliary functions to perform the translation. Some of these implement the analysis phase.

Figure 5.22 shows the main translation function *TransSCJProg*, which accepts an SCJ program and an *Identifier*, which is the SCJ program's name. This function calls a number of others, which build and generate specific parts of the model. Table 5.9 shows the functions used by *TransSCJProg*, the phase to which they belong, and the component they produce. Each build function produces an environment that holds the information needed to generate the model of that particular component. For example, the *BuildFWEnv* function produces a *FWEnv*, which is used by the *GenerateFWProcs* function to generate the framework processes. In addition to the functions in Table 5.9, the *TransClasses* function translates the application classes. The definitions of the functions omitted here can be found in Appendix E.

The *Program* process is defined inside the *TransSCJProg* function (Fig. 5.22) and takes the form **process** $Program \mathrel{\widehat=} (Framework \llbracket appSync \rrbracket Bound\_Application) \llbracket lockSync \rrbracket Locking$, as explained in the previous section. The *framework* type is defined, outside of *TransSCJProg*, as a sequence of all the framework processes. The result of *TransSCJProg* is defined as the concatenation of the *framework* type, the translated application processes *app*, the *MethodCallBinder* processes *mcbModel*, the *Locking* processes *lockModel*, the *Program* process, and two processes that are constructed during the result's definition. The first process is $\langle procDef(pd(FWName, head\, fwProcs))\rangle$, which defines the *Framework* process in our z BNF encoding of *Circus*. The second is $\langle procDef(pd(AppName, appProc))\rangle$, which defines the *Bound_Application* process. This sequence of processes corresponds to the processes described in Sect. 5.1 combined with the framework model in Chap. 4.

To illustrate how components of the model are translated by *TransSCJProg* using other functions, we examine the translations of the framework processes. Figure 5.23 shows the build phase function *BuildFWEnv*, which takes an *SCJProgram* and extracts the information required to generate the framework processes. It produces an *FWEnv* (defined in Fig. 5.24), which holds the information extracted from the program. The *FWEnv* type is a schema, which records the identifier of the top-level mission sequencer and a non-empty sequence of *TierEnv*s. Note that an *FWEnv* does not need to hold any information about the safelet because its framework process does not take any parameters.

The *BuildFWEnv* function extracts the top-level mission sequencer from the SCJ program

$$
\begin{array}{l}
\mathit{TransSCJProg} : \mathit{Identifier} \times \mathit{SCJProgram} \pfun \mathit{CircusProgram} \\
\hline
\mathit{dom}\ \mathit{TransSCJProg} = \mathit{Identifier} \times \mathit{TranslatablePrograms} \\
\forall\, \mathit{name} : \mathit{Identifier};\ \mathit{scjProg} : \mathit{SCJProgram} \\
\quad \bullet \exists\, \mathit{app} : \mathit{CircusProgram};\ \mathit{program} : \mathit{CircusProgram}; \\
\qquad \mathit{fwProcs} : \operatorname{seq} \mathit{Process};\ \mathit{appProc} : \mathit{Process};\ \mathit{lockModel} : \operatorname{seq} \mathit{CircusParagraph}; \\
\qquad \mathit{mcbModel} : \operatorname{seq} \mathit{CircusParagraph};\ \mathit{fwEnv} : \mathit{FWEnv}; \\
\qquad \mathit{appEnv} : \mathit{AppProcEnv};\ \mathit{mcbEnvs} : \mathit{MCBEnv};\ \mathit{lockEnv} : \mathit{LockingEnv} \mid \\
\qquad\quad \mathit{fwEnv} = \mathit{BuildFWEnv}(\mathit{scjProg}) \\
\qquad\quad \mathit{appEnv} = \mathit{BuildAppProcEnv}(\mathit{scjProg}) \\
\qquad\quad \mathit{mcbEnvs} = \mathit{BuildMCBEnvs}(\mathit{scjProg}) \\
\qquad\quad \mathit{lockEnv} = \mathit{BuildLockEnv}(\mathit{scjProg}) \\
\qquad\quad \mathit{app} = \mathit{TransClasses}(\mathit{scjProg}) \\
\qquad\quad \land\ \mathit{fwProcs} = \mathit{GenerateFWProcs}(\mathit{fwEnv}) \\
\qquad\quad \land\ \mathit{appProc} = \mathit{GenerateAppProc}(\mathit{appEnv}) \\
\qquad\quad \land\ \mathit{mcbModel} = \mathit{GenerateMCBModel}(\mathit{mcbEnvs}) \\
\qquad\quad \land\ \mathit{lockModel} = \mathit{GenerateLockModel}(\mathit{lockEnv}) \\
\qquad\quad \land\ \mathit{program} = \langle \mathit{procDef}(\mathit{pd}(\mathit{ProgName}(\mathit{name}), \\
\qquad\quad \mathit{procHide}(\mathit{procPar}(\\
\qquad\qquad \mathit{procHide}(\\
\qquad\qquad\quad \mathit{procPar}(\\
\qquad\qquad\qquad \mathit{procName}(\mathit{FWName}), \\
\qquad\qquad\qquad \mathit{appComms}, \\
\qquad\qquad\qquad \mathit{procHide}(\\
\qquad\qquad\qquad\quad \mathit{procPar}(\mathit{procName}(\mathit{AppName}), \\
\qquad\qquad\qquad\quad \mathit{mcbComms}, \\
\qquad\qquad\qquad\quad \mathit{procName}(\mathit{MCBName})), \\
\qquad\qquad\qquad \mathit{mcbComms})), \\
\qquad\qquad\quad \mathit{appComms}), \\
\qquad\qquad \mathit{lockComms}, \\
\qquad\qquad \mathit{procName}(\mathit{LockName})), \\
\qquad\quad \mathit{lockComms})))\rangle \bullet \\
\qquad\quad \mathit{TransSCJProg}(\mathit{name}, \mathit{scjProg}) = \\
\qquad\qquad \mathit{framework} \cat \langle \mathit{procDef}(\mathit{pd}(\mathit{FWName}, \mathit{head}\, \mathit{fwProcs}))\rangle \\
\qquad\qquad\quad \cat \mathit{app} \cat \langle \mathit{procDef}(\mathit{pd}(\mathit{AppName}, \mathit{appProc}))\rangle \\
\qquad\qquad\quad \cat \mathit{mcbModel} \cat \mathit{lockModel} \cat \mathit{program}
\end{array}
$$

Figure 5.22: The Top-Level Translation Rule *TransSCJProg*

$$
\begin{array}{|l}
BuildFWEnv : SCJProgram \pfun FWEnv \\
\hline
dom\ BuildFWEnv = TranslatablePrograms \\
\forall scjProg : dom\ BuildFWEnv \\
\quad \bullet \exists tlmsID : Identifier;\ tlmsBody : MissionSequencerClassBody; \\
\qquad tiers : \mathrm{seq}\ Tier \mid \\
\qquad\quad ProgTLMS(scjProg) \neq NoSequencer \\
\qquad\qquad \Rightarrow ProgTLMS(scjProg) = tlms(tlmsID, tlmsBody) \\
\qquad\quad \bullet BuildFWEnv(scjProg) = \\
\qquad\qquad \langle\!|\, TopLevelMS == tlmsID, Tiers == BuildTierEnvs(ProgTiers(scjProg)) \,|\!\rangle
\end{array}
$$

Figure 5.23: The *BuildFWEnv* Function

$$
\begin{array}{|l}
\text{\_\_ } FWEnv \text{ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
TopLevelMS : Identifier \\
Tiers : \mathrm{seq}\ TierEnv \\
\hline
Tiers \neq \langle\rangle \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\text{\_\_ } TierEnv \text{ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
Clusters : \mathrm{seq}\ ClusterEnv \\
\hline
Clusters \neq \langle\rangle \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\text{\_\_ } ClusterEnv \text{ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
Mission : Identifier \\
NestedMissionSequencers : \mathbb{P}\ Identifier \\
ManagedThreads : \mathbb{P}\ Identifier \\
PeriodicEventHandlers : \mathbb{P}\ Identifier \\
AperiodicEventHandlers : \mathbb{P}\ Identifier \\
OneShotEventHandlers : \mathbb{P}\ Identifier \\
\hline
disjoint \langle NestedMissionSequencers, ManagedThreads, PeriodicEventHandlers, \\
\quad AperiodicEventHandlers, OneShotEventHandlers \rangle \\
\quad \bigcup\{NestedMissionSequencers, ManagedThreads, PeriodicEventHandlers, \\
\quad AperiodicEventHandlers, OneShotEventHandlers\} \neq \varnothing \\
\hline
\end{array}
$$

Figure 5.24: The Environments for the Framework processes

Function	Parameters	Result
BuildTierEnvs	seq *Tier*	seq *TierEnv*
BuildTierEnvs	*Tier*	*TierEnv*
BuildClusterEnvs	seq *Cluster*	seq *ClusterEnv*
BuildClusterEnv	*Cluster*	*ClusterEnv*
BuildSOEnvs	$\mathbb{F}$ *SchedulableObject*	$\mathbb{F}$ *Identifier* × $\mathbb{F}$ *Identifier* × $\mathbb{F}$ *Identifier* × $\mathbb{F}$ *Identifier* × $\mathbb{F}$ *Identifier*
GetIdentifiers	$\mathbb{F}$ *SchedulableObject*	$\mathbb{F}$ *Identifier*

Table 5.10: Functions Used by *BuildFWEnv* to Build Ttier Environments

Function	Parameters	Result
TierSync	*TierEnv*	*CSExpression*
GenerateTierFWProcs	*FWEnv*	seq *Process*
GenerateTierFWProc	*ClusterEnv*	*Process*
GetParams	*Identifier*	seq *Expression*

Table 5.11: Functions Used by the *GenerateFWProcs* Function

using the *ProgTLMS* function. *BuildFWEnv* only extracts the top-level mission sequencer if it is not equal to *NoSequencer*, which is the value that represents the safelet returning `null` from its `getSequencer()` method. The *ProgTLMS* function extracts a value of top-level mission sequencer type, where its identifier (*tlmsID*) is the identifier of the top-level mission sequencer.

BuildFWEnv uses a chain of functions, summarised in Table 5.10, to build the environments for the tiers. First, the *BuildTierEnvs* function, which takes a sequence of program tiers and produces a sequence of *TierEnv*s. We use *ProgTiers* to extract the sequence of tiers from the SCJ program. The *BuildTierEnvs* function uses the *BuildTierEnv* function to build the environment of each tier, which in turn uses *BuildClusterEnvs* to build the environments for each cluster. The *BuildClusterEnvs* function calls *BuildClusterEnv* to build the environment for each cluster, which in turn calls *BuildSOEnvs* to build the environments for the schedulable objects in that cluster. Finally, *BuildSOEnvs* uses the *GetIdentifiers* function to extract the identifiers from the sequence of schedulables objects in that cluster.

Figure 5.25 shows the *GenerateFWProcs* function, which generates the processes that compose the *Framework* process. This implements the first step of stage one in the generate phase (described Sect. 5.1.3). The function takes an *FWEnv* environment and produces a sequence of *Process* values, each of which defines a *Circus* process.

Table 5.11 shows the functions used by the *GenerateFWProcs* function. The *TierSync* function takes two tier environments and from these generates a channel set that allows the two tiers to communicate. The order of the *TierEnv* parameters must be the same as the order of the tiers that they represent. For example, the environments representing the highest tier in a program's hierarchy and the next tier down, *Tier*0 and *Tier*1 respectively, should be passed in that order so that *TierSync* can produce a channel set that allows them to

```
GenerateFWProcs : FWEnv → seq Process
─────────
∀ env : FWEnv
  • ∃ fwProc : Process; controlTierProc : Process; tierProcs : seq Process
    | fwProc = procPar(
        procName(ControlTier),
        TierSync(head env.Tiers),
        GenerateTierFWProc(env.Tiers)
    )
    ∧ controlTierProc = procPar(
        procName(SafeletFWName),
        ControlTierSync,
        procInstP(procName(TopLevelMissionSequencerFWName),
        GetParams(env.TopLevelMS))
    )
    ∧ tierProcs = GenerateTierFWProcs(env.Tiers)
    • GenerateFWProcs(env) = ⟨fwProc⟩ ⁀ ⟨controlTierProc⟩ ⁀ tierProcs
```

Figure 5.25: The *GenerateFWProcs* Function

communicate.

To produce the channel set we extract the mission identifiers (and possibly mission sequencer identifiers) from the second *TierEnv* parameter, which represents the lower of the two tiers. These identifiers are used to restrict some of the channels in the set, so that the synchronisation only occurs for the events intended to communicate with processes in these two tiers. The *GenerateTierFWProcs* function generates a sequence of tier processes using the *GenerateTierFWProc* function to generate each individual process. Finally, the *GetParams* function extracts from an environment the API-parameters to the object it represents, which become parameters to the process. This captures, for example, the parameter classes passed to schedulables to specify their deadline, deadline miss handler, and so on.

This approach is reused to produce the application process, method call binder, and the locking model. Each of the components shown in Table 5.9 is translated by a a generate function, which constructs the component based on the information in an environment produced by a build function. These functions also make use of various auxiliary functions to help the translation.

The full listing of the translation functions, including those presented in this section, is presented in Appendix E. It covers 43 functions and several other schemas, types, and other definitions. It has been parsed and type checked using the Community Z Tools [65]. The structure of this formalisation mirrors that of the translation strategy (presented in Sect. 5.1), and is mirrored by the automatic translation tool T$^{\text{ight}}$R$^{\underline{\text{ope}}}$ presented in Sect. 5.3, below. Figure 5.4 shows the environments that are produced by the build phase. These environments are implemented in the automatic translation tool, which is described in the next section.

5.3 Automatic Translation

In this section we present T$^{\text{ight}}$R$^{\underline{\text{ope}}}$, a Linux Java prototype tool for automatic translation of SCJ Level 2 programs into *Circus*. It, like our modelling approach, is based on the Level 1 translation tool, TransCircus, in [93]. It reuses the structure and method body translation technique found in TransCircus, but in all other aspects T$^{\text{ight}}$R$^{\underline{\text{ope}}}$ is vastly re-engineered.

TransCircus comprises 122 classes over approximately 9100 lines of code. In contrast, T$^{\text{ight}}$R$^{\underline{\text{ope}}}$ comprises 48 classes and has approximately 12500 lines of code. Once the location of the input program has been identified the translation and output is fully automatic. T$^{\text{ight}}$R$^{\underline{\text{ope}}}$ reuses some classes from TransCircus, as described in Sect. 5.3.1. However, it only reuses one class from the package that captures application specific information from Level 1 programs (`tool.modelgen`) because our new model requires new classes to capture Level 2 programs.

TransCircus requires its input programs to be annotated with information including the class's identifier and if it requires a process to model it or just an *OhCircus* class. By contrast, T$^{\text{ight}}$R$^{\underline{\text{ope}}}$ extracts all the required information from the program, without annotations.

5.3.1 Overview

The translation performed by T$^{\text{ight}}$R$^{\underline{\text{ope}}}$ implements the same three phases as described in Sect. 5.1. First, the analysis phase compiles the input program to produce a list of Abstract Syntax Trees (AST), then extracts information useful for the translation from the ASTs during a pre-processing step. The build phase constructs environment objects from the input ASTs, using the information extracted by the analysis phase. Finally, the generate phase uses the information in the environments to produce the output model files. This is achieved by combining each environment with a template, which dictates the shape of the process, replacing tags in the template with the relevant information from the environment.

T$^{\text{ight}}$R$^{\underline{\text{ope}}}$ can translate all the unique features of SCJ Level 2. It requires that the SCJ program compiles and is structured with one paradigm class per file. It also requires that programs conform to an input pattern, which simplifies statements without altering their semantics. Complex Java statements must be rewritten: chains of method calls in one statement become several separate method calls, and if a method parameter is an new object, then any parameters the object takes must be literal values. Finally, the condition for a while loop must not be a variable, the entire condition should be contained within the while loop statement. Note that these are restrictions of the current version of the tool, not of the technique as a whole.

Figure 5.26 shows a package diagram of T$^{\text{ight}}$R$^{\underline{\text{ope}}}$. There are five `tools.tightrope` packages (in bold), which compose the core of the application, and six smaller `tools` packages, which are reused from TransCircus [93]. The `tools.application` package contains the main classes of both TransCircus and T$^{\text{ight}}$R$^{\underline{\text{ope}}}$. The `tools.application.TightRope` class is the entry point of the application and controls the program flow.

The `tools.analysis` and `tools.compiler` packages are reused from TransCircus by the analysis phase to compile the SCJ program (Sect. 5.3.2). As previously mentioned, the

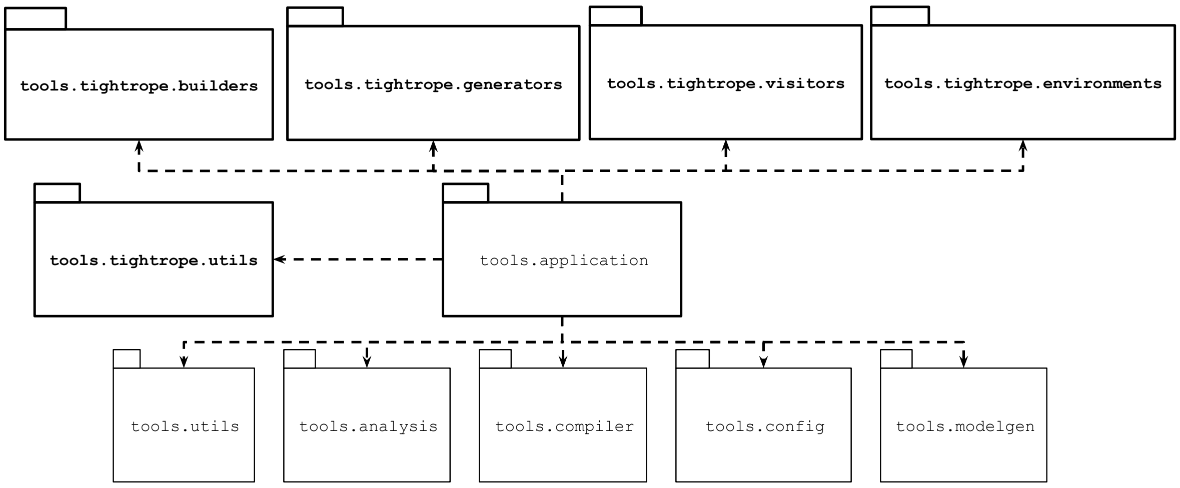


Figure 5.26: Package Diagram of T^{ight}R^{ope}

`tools.modelgen` package provides the classes that perform the translation in TransCircus. TightRope reuses one class form this package, which represents a parsed SCJ program. The build phase (Sect. 5.3.3) uses the classes in three packages, `tools.tightrope.builders`, `tools.tightrope.environments`, and `tools.tightrope.visitors` to translate the SCJ program and store the translation in environments. The `tools.tightrope.generators` package is used by the generate phase, described in Sect. 5.3.4, to produce the model from the information in the environments. The `tools.tightrope.utils` and `tool.utils` packages provide utility classes that support the translation. Finally, TightRope uses one class from the `tools.config` package to provide access to the properties in the tool's configuration file.

5.3.2 Analysis Phase

The analysis phase occurs in three steps. In the first step, we take the input program and compile it into a list of ASTs representing the program's classes, which facilitates the translation. The program is compiled using the Java `Compiler Tree` API[1] against the Icecap SCJ implementation [46], but any valid Level 2 implementation can be used.

In the second step, we iterate through the list of ASTs produced by the first step and preprocess them to build two of the maps described in Sect. 5.1.1. The first map, `classTypeMap`, maps each class in the SCJ program to a list of the names of its supertypes. For each AST, we extract the interfaces it implements and the class it extends. A list of these components and the name of the class represented by the AST are added to the `classTypeMap`. The generate phase uses the `classTypeMap` to identify the output template for each class, using the rules described in Table 5.1.

The second map, `classMethodsMap`, maps each class to a list of the names of its methods that are not defined in the SCJ API. For each AST, we iterate through its methods. For each method, if it is not defined in the SCJ API, then we add it's name to a temporary list. Once all the methods have been checked, if the temporary list is not empty, we add the name of the class that the AST represents and the temporary list to the map. The `classMethodsMap` is used to help construct the *MethodCallBinder* process. The analysis phase (in Sect. 5.1.1)

[1]`docs.oracle.com/javase/7/docs/jdk/api/javac/tree/`

Class	Environment
`ProgramEnv`	*NetworkEnv*, *Threads*, *Objects*
`StructureEnv`	*ProgramEnv*, *TierEnv*, *ClusterEnv*, *SchedulableEnv*
`PeriodicEventHandlerEnv`	*PEHEnv*
`AperiodicEventHandlerEnv`	*APEHEnv*
`OneShotEventHandlerEnv`	*OSEHEnv*
`ManagedThreadEnv`	*MTEnv*
`MissionEnv`	*MissionEnv*
`MissionSequencerEnv`	*MissionSequencerEnv*
`SafeletEnv`	*SafeletEnv*
`ClassEnv`	*ClassEnv*
`MethodEnv`	*MethodEnv*
`VariableEnv`	*VarEnv*

Table 5.12: Table Mapping the Builder Classes to the Environments they Implement

produces a third map, which maps method names to the names of classes that call that method. In T$^{\text{ight}}$R$^{\text{ope}}$ this map is constructed during the build phase, which we describe in Sect 5.3.3.

Finally, the third step produces a list of all relevant non-paradigm objects in the program. This list includes any object that does not implement the `Safelet` interface or extend the `MissionSequencer`, `Mission`, or one of the schedulable classes. The Icecap implementation includes a `Launcher` class to act as the entry point to the SCJ program. Since this is not in the API or part of the program, it is also excluded from this list.

5.3.3 Build Phase

The build phase takes the ASTs and maps from the analysis phase and constructs environment objects (contained in the `tools.tightrope.environments` package) that capture the information in the system using classes in the `tool.tightrope.builder` package. The environment classes, shown in Fig. 5.27, are the implementation of the environments shown in Fig. 5.4. The builder classes, shown in Fig. 5.28, are the implementation of the processes used to build each type of environment, described in Sect. 5.1.2. Table 5.12 shows which functions and types are implemented by these classes.

Each type of object has its own specialised environment that holds the information needed by the generate phase to produce its model. The environments for each paradigm object – the safelet, mission sequencers, missions, and schedulables – extend the `ParadigmEnv` class. Non-paradigm objects, those that do not extend any of the paradigm objects, are represented by the `NonParadigmEnv` environment. Both the `NonParadigmEnv` and `ParadigmEnv` classes extend the `ObjectEnv` class, which is the superclass that characterises all the environments. The `ProgramEnv` environment records information such as the identifiers in the program, the non-paradigm objects, and the program's structure. The `StructureEnv` class records the structure of the program and the environments for each of the paradigm objects.

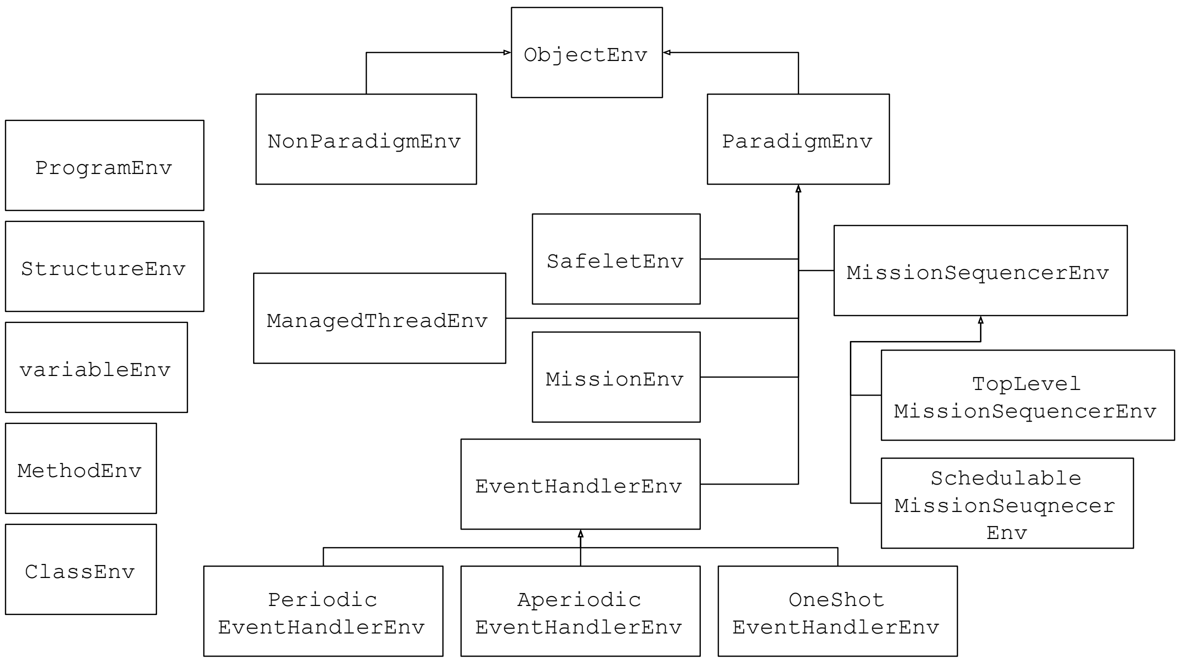


Figure 5.27: Class Diagram of the `tools.tightrope.environments` Package

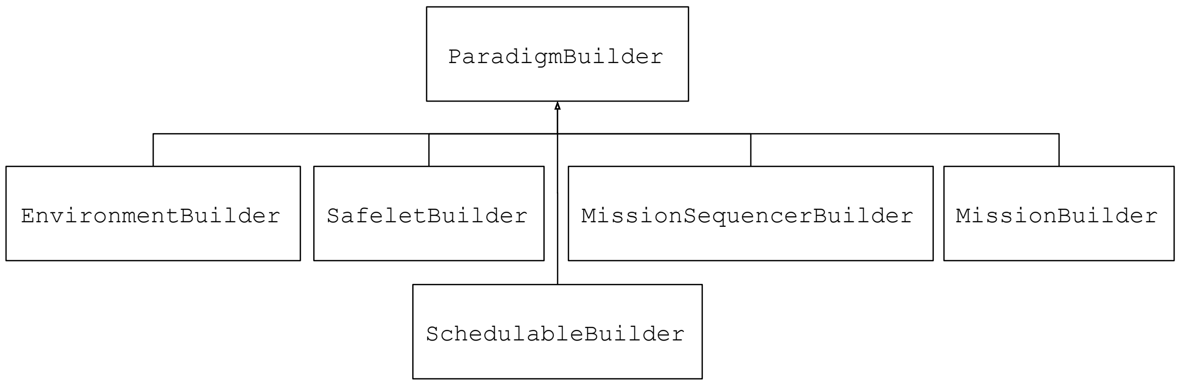


Figure 5.28: Class Diagram of the `tools.tightrope.builders` Package

This phase operates in the same way as illustrated in Fig. 5.3. First, the environments of the non-paradigm objects are built, because they do not affect the structure of the program. The preprocessing that occurs in the analysis phase identifies the safelet, and it is the first paradigm object to be translated because it is at the top of the program's hierarchy. The mission sequencer returned by the safelet's `getSequencer()` method is the top-level mission sequencer, and its environment is built next. After the top-level mission sequencer, we build the environment of each mission returned by its `getNextMission()` method. For each of these missions, we build the schedulable objects registered to it, and then move on to the next mission. If a mission registers a schedulable mission sequencer, then we build the environments of each of that mission sequencer's mission and their registered schedulables before moving on to the next schedulable. Once there are no more missions to translate that are controlled by the top-level mission sequencer, we move on to the generate phase. Note that, for simplicity, TightRope assumes that there will only be one top-level mission sequencer, whereas the translation strategy described in Sect. 5.1 accepts the possibility of multiple top-level mission sequencers.

The environment of each type of object is built by a specific builder class, each of which extends a generic builder class `ParadigmBuilder` that contains methods reused by all the builder classes. For example, the `SafeletBuilder` class, which extends `ParadigmBuilder`, builds the environment of the program's safelet.

Each builder class has a `build()` method, which traverses the AST of an object and extracts its variables and methods. Because this functionality is required for any builder class, it is encapsulated in the classes in the `visitors` package. These visitor classes implement the process of building an environment, which is described in Sect. 5.1.2. In addition to the generic behaviour, the `build()` method of each builder class is specialised to extract the contents of the overridden API methods.

Each visitor traverses the tree of a particular component to retrieve the information required for its environment. The visitor classes do not have as obvious a correspondence to the functions in the core formalisation of the translation. This is stylistic; each visitor is specialised to the SCJ component from which it is extracting information (methods or method bodies, for example) as opposed to the model component for which it is producing the information.

The `MethodVisitor` class traverses a tree representing a method and translates its name and parameters, and uses other visitors to extract the return type and value, and body. It produces a `MethodEnv` containing this information. The `ReturnVisitor` class traverses a tree representing a method and extracts a list of the names that are returned by that method. The `RegistersVisitor` class traverses a tree representing the `Mission.initialize()` method and extracts the name of the schedulables on which `register()` is called.

The `MethodBodyVisitor` class traverses a tree representing a method and translates its body using the rules described in Sect. 5.1.2. The `VariableVisitor` class traverses a tree representing a member of a class and extracts any variables that tree might contain. This class corresponds to the translation rules presented in Table 5.3. Finally, the `ParametersVisitor` class is used to translate method and class parameters. It traverses a tree representing an expression and extracts the name, type, and value of the parameter.

The `SafeletBuilder.build()` method builds the environment of a safelet object. In addition to the object's variables and methods, it specifically captures two extra components. First, it translates the safelet's `initializeApplication()` method and stores it in the `SafeletEnv`. Then, it translates the names of the mission sequencers returned by the `getSequencer()` method. This is achieved by the `ReturnVisitor`, as described above. The mission sequencer identified here is the top-level mission sequencer and identifies the next component to be built.

The `MissionSequencerBuilder` class builds the environment of a mission sequencer object. It is used to build the environments of both top-level and schedulable mission sequencers. In addition to the object's variables and methods, its `build()` method specifically captures the missions returned by the `getNextMission()` method. Again, this is achieved by the `ReturnVistor` class, as described above. These missions are stored in the `MissionEnv` and identifies the next components to be built.

The `MissionBuilder` class builds the environment of a mission object. In addition to the objects variables and methods, its `build()` method specifically captures two extra components. First, it translates the `cleanUp()` method. Then, it translates the schedulables that are registered during its `initialize()` method. This is achieved by the `RegistersVisitor`, which finds a call to the `register` method of a schedulable object and returns the name of that schedulable. The schedulables identified are stored in the `MissionEnv` and identifies the next components to be built.

The `SchedulableBuilder` class builds the environment of any schedulable object. It is used to build the environments of any of the event handler classes and the managed thread class. In addition to the object's variables and methods, its `build()` method translates the `handleAsyncEvent()` method of an event handler or the `run()` method of a managed thread. These methods are translated by the `MethodVisitor`, as described above. The `build()` method also checks if an aperiodic event handler's `handleAsyncEvent()` method takes a `long` parameters or not, and records it in the environment. This information is required because we model the behaviour of these two types of aperiodic event handler differently.

5.3.4 Generate Phase

The generate phase produces the models of the program by combining a template with an environment, which has been produced by the build phase. There are different templates for processes, *OhCircus* classes, and channel sets. The combination process replaces the tags in the template with the relevant application-specific information from the environment. For example, each process template uses the name of the class to generate the name of the process. So the combination of a process template with an environment replaces the tag for the process name with the actual name of the class in the environment.

The `CircusGenerator` class controls the generate phase, using its `generate()` method, which performs the combination of templates and environments using the Freemarker template engine. The templates used in this phase are implementations of the templates, presented in Sect. 5.1, for Freemarker.

This phase implements the two stages of the generate phase described in Sect. 5.1.3. As previously mentioned, stage one produces the top-level network of processes that control our model. It is implemented by the `generateNetwork()` method. Stage two produces the *Circus* processes, *OhCircus* classes, and channels modelling the objects in the program. The processes and channels for each type of object are generated by a different generate method.

The `generateNetwork()` method uses the information in the `ProgramEnv` to generate the *Framework*, *Bound_Application*, *MethodCallBinder*, and *Locking* processes. It also generates the channels that these processes require. The order in which these steps occur is unimportant, because the structure of the program has already been captured by the build phase. Each of these components is generated from the information in the `ProgramEnv`, but uses a different template. These templates are the same as described in stage one of the generate phase in Sect. 5.1.3, but implemented for Freemarker.

The other generate methods compose stage two. They generate the non-paradigm objects and then the safelet, the top-level mission sequencer, missions, and schedulables. Again, the order in which these components are generated is unimportant because the program structure has been captured during the build phase. Each of these `generate` methods takes the information from a particular type of environment and combines it with a particular template. For example, the `generateSafelet()` method takes the information in a `SafeletEnv` and combines it with the safelet template to produce the *SafeletApp* process.

In addition to generating the model, the `generate()` method performs some tasks that are specific to T$^{\underline{\text{ight}}}R^{\underline{\text{ope}}}$. It writes the files of the model to an output folder. It also generates a report of the translated program that contains all the application-specific processes, *OhCircus* classes, and channels. The report is written in LaTeX, and T$^{\underline{\text{ight}}}R^{\underline{\text{ope}}}$ compiles the report after the generate phase has completed.

5.4 Translation Examples

Our translation strategy, presented in Sect. 5.1, captures the application-specific behaviour of SCJ Level 2 programs. In particular, we model managed threads, schedulable mission sequencers, and suspension, as these are unique features of Level 2. We used the example applications[2] summarised in Table 5.13 to test the translation strategy during its development. These examples exhibit a range of features available in SCJ Level 2 programs, including: single and sequential missions, nested mission sequencers, all of the types of schedulable object, and programs with several tiers to test the termination protocol on more complex program structures. The final two applications are the running examples that we have previously described, Buffer in Sect. 2.1.2 and Aircraft in Sect. 2.1.3.

During the development of the translation strategy, the example applications have highlighted particular problems with our model and allowed us to fix them. For example, the Mission2 example illuminated that a managed thread terminating before it is requested to (by its `run()` method returning) produced a spurious deadlock. This occurred when the controlling mission attempted to request the managed thread to terminate, but the managed thread was no longer offering the termination request event. Managed threads are the only schedulable capable of terminating without being requested to, and the Mission2 example

[2]The example SCJ programs and their translations are available at `http://www.cs.york.ac.uk/circus/hijac/case.html`

Name	Description	№ Classes	Translation Time (s)
Mission1	A single mission with periodic event handler that releases an aperiodic event handler	5	∼1.53
Mission2	A single mission with a managed thread and a one-shot event handler	5	∼1.27
ThreeOneShots	A single mission with three one-shot event handlers	6	∼1.28
ThreeThreads	A single mission with three managed threads	6	∼1.21
SequentialMissions	Two sequential missions, each with two managed threads	8	∼1.28
NestedSequencer1	A single mission with a single nested mission sequencer	7	∼1.21
NestedSequencer2	A mission, with three nested mission sequencers. Each has one mission controlling a periodic event handler	14	∼1.34
NestedSequencer3	A mission, with a nested mission sequencer that has two sequential nested missions, each with a managed thread.	8	∼1.30
NestedSequencer4	A complicated example using two levels of nesting. It contains 4 missions and 3 managed threads	12	∼1.18
NestedSequencer5	Extends NestedSequencer4, combines complex nesting, all schedulable types, and sequential missions	12	∼1.28
Buffer	Small program using managed threads and synchronisation	6	∼1.23
Aircraft	A multiple-mode program using a schedulable mission sequencer to represent phases of aircraft flight	23	∼2.57

Table 5.13: Summary of SCJ Translated Example Applications

Name	№ Classes	States	Assertion Time (s)		
			Compilation	Checking	Total
Mission1	5	549	0.25	0.39	0.64
Mission2	5	138	0.42	0.10	0.52
ThreeOneShots	6	1,460	0.20	0.41	0.61
ThreeThreads	6	213	0.18	0.33	0.51
SequentialMissions	8	343	0.27	0.40	0.67
NestedSequencer1	7	147	0.64	0.12	0.76
NestedSequencer2	14	898,584	4.57	3.65	8.22
NestedSequencer3	8	311	0.98	0.13	1.11
NestedSequencer4	12	6,417	4.13	0.24	4.37
NestedSequencer5	12	36,219	6.76	0.64	7.40
Buffer	6	310	0.41	1.02	1.43
Aircraft	23	∼ 46,647	∼ 20.75	∼ 59.41	∼ 80.16

Table 5.14: Summary of Model States

helped us to model this behaviour correctly.

The NestedSequencer2 example exposed a problem with the periodic event handler framework process when it requests its controlling mission to terminate. The resulting attempt by the mission to terminate the periodic event handler caused the deadlock, which this example helped us to resolve. Further, the NestedSequencer4 and NestedSequencer5 examples are particularly useful illustrations of the complex structures that nested mission sequencers in Level 2 programs allow. These two examples also provided a test bed for modelling the new termination protocol that we proposed in [56], as applied to a program with several tiers. This shows the termination request 'bubbling' up the program hierarchy.

T$^{\text{ight}}$R$^{\text{ope}}$, described in Sect. 5.3, implements the translation strategy (Sect. 5.1) and provides automatic translation of SCJ programs to *Circus*. It has been used to translate all of the example applications summarised in Table 5.13, where the reported translation times are from running T$^{\text{ight}}$R$^{\text{ope}}$ on a Lenovo W540 with an Intel Core i7-4700MQ CPU. All of the example programs were translated in less than 3 seconds. The Buffer and Aircraft are the most complex applications translated, each exhibiting a unique feature of SCJ Level 2. Buffer contains six classes, two of which are managed threads that use synchronisation and suspension to share access to a bounded buffer held by their controlling mission. This application was translated in ∼1.23 seconds. Buffer was the test bed application for the *ObjectFW* and *ThreadFW* processes (which control the synchronisation behaviour) and how they interact with the rest of the model. Aircraft contains 23 classes in a more complex hierarchy than the Buffer application. Aircraft uses a schedulable mission sequencer and both aperiodic and periodic event handlers. The Aircraft example was translated in ∼2.57 seconds. While this is a small sample of examples, they lend us confidence that our automatic translation can deal with structurally complex programs and will scale well.

The *Circus* models of the programs in Table 5.13 have been validated by translating them to CSP$_M$, which is the machine-readable version of CSP. The framework model was also translated to CSP$_M$. The CSP version of the models have been analysed using the Failures Divergences Refinement tool (FDR3) [33]. This analysis includes animating the framework model to compare its behaviour to that described in the SCJ Language Specification, and both animating and model checking the models of full programs to check their behaviour against running programs and to check for deadlock and divergence freedom.

Table 5.14 summarises, for the CSP$_M$ version of each of the modelled programs, the number of states and the time taken (in seconds) to check a deadlock freedom assertion in FDR3. The assertion times shown are split into the compilation and checking phases of the FDR3 assertion check, and their sum. The use of particular processes and combinations of processes increases the number of states. For example, the large number of states in ThreeOneShots application is due to the possibility that a `OneShotEventHandler` may be descheduled and rescheduled.

The model of the NestedSequencer5 application has the third highest number of states because the program has three tiers and one of each type of schedulable, which produces more states because of the termination-related events. As Table 5.14 shows, the model takes longer to compile (6.76s) than it does to perform the deadlock check (0.64s). This is because of the large number of events in the model, because of the communication between the tiers, which increases the compilation time. The model of the NestedSequencer2 application has the highest number of states because it contains two tiers and three `PeriodicEventHandler` processes. In general, models containing *PeriodicEventHandler* processes have more states because of the possibility of period overruns.

Despite using synchronisation and suspension, the model of the Buffer application contains only 310 states. The processes controlling synchronisation and suspension in the *Circus* version of the model contain a lot of variables, so extra attention was paid when they were translated to CSP$_M$. This has resulted in a version of the synchronisation and suspension modal that does not unnecessarily inflate the state space of the model and is better suited to analysis in FDR. The compilation time of the model (0.41s) is faster than its checking time (1.02s) because it has a large number of states, due to the model's complexity as opposed to large number of events

The translation from *Circus* to CSP$_M$ is relatively straightforward, because CSP is part of *Circus*. We translate each *Circus* process and its actions into a CSP process comprising processes representing the *Circus* actions. This provides a CSP model that mirrors the structure of the *Circus* process, but which lacks the encapsulation provided by *Circus*.

Because CSP lacks variables, the state component of a *Circus* process is translated into a CSP process that is parametrised by values representing the *Circus* variables. This 'state process' provides channels to 'read' and 'write' to the parameters it controls, before recurring (possibly with an updated value) to offer the 'read' and 'write' channels again.

The translation of state from *Circus* to CSP$_M$ occasionally produced models that FDR3 could not analyse, because of state explosion. Firstly, each *Circus* variable requires (usually at least two) extra channels in the CSP model. Secondly, and more importantly, state processes can yield a large number of model states. This is especially true when one of its parameters is a set or a sequence. The CSP$_M$ version of the framework model has been refactored to improve the tractability of analysis in FDR3, by reducing the number of states while maintaining the same behaviour[3].

For example, the previously mentioned synchronisation and suspension processes were a particular focus for refactoring, as the original translation from *Circus* proved intractable because of the large number of variables. We reduced the use of variables and reorganised the processes to ensure a smaller state space for the synchronisation and suspension system. Other refactorings ranged from simple restrictions, such as defining a sequential order for the termination of a mission's schedulables instead of allowing them to interleave, to defining a process to capture a complicated data structure. An example of the latter is the priority queue (Fig. 4.6), used to ensure the highest priority thread that has also been waiting the longest gains a lock, which is a function in the *Circus* version of the model. The FDR-friendly translation to CSP$_M$ defines this as an parallelism of small processes, which each control one element of the queue. These processes toggle a cell between being empty and being full, and specify the events available in each of these two states. This proves a much more tractable approach for checking in FDR.

The model of the Aircraft application is the biggest that we have checked, having 23 classes arranged over two tiers. It is also relatively complex because the schedulables in both tiers access variables held in the main mission (in Tier 0), which generates a lot of communication between the tiers. Because of this complexity, the Aircraft model required modifications that reduced the state space to allow it to be checked – hence the results in Table 5.14 being approximations. This model seems to be at the limit of what our verification technique can handle, with the current CSP$_M$ version of our model. We not that this is not a limitation of the modelling approach as a whole. Further work on improving the efficiency of our CSP$_M$ models, especially when translated from *Circus*, will improve the scalability of the verification technique.

5.5 Summary

This chapter presents a detailed description of our strategy for translating SCJ Level 2 programs into *Circus*. This strategy captures the application-specific behaviour of SCJ programs and produces *Circus* models that are compatible with the model of the SCJ API described in Chap. 4. The combination of these two models is also captured by the translation strategy, and this chapter presents the information and rules required to perform this combination.

This chapter also presents two implementations of the translation strategy. The first is a formalisation in Z of the core elements of our translation, which paves the way for a full formalisation that could be used to prove the soundness of our technique. The second is the implementation of T$^{\text{ight}}$R$^{\underline{\text{ope}}}$, a tool for automatic translation.

To test the translation strategy, we have translated 12 SCJ example applications. These programs cover the full-range of Level 2 features, including managed threads, suspension, and schedulable mission sequencers. These example applications include the Buffer in Sect. 2.1.2 and Aircraft in Sect. 2.1.3. These examples have been translated by hand and by our translation tool, T$^{\text{ight}}$R$^{\underline{\text{ope}}}$. However, the tool places some limitations on the input programs it can accept. It is important to note that these are simplifications required by the tool only,

[3]This work was done with the help of Tom Gibson-Robinson, University of Oxford, UK

and that they are not limitations of the technique as a whole. These limitations add a small programmer overhead, as they require some program rewriting, but do not restrict the functionality of the programs $\mathrm{T^{ight}R^{\underline{ope}}}$ can handle.

To validate the translated *Circus* models, they have been translated to CSP_M and analysed using FDR3. The analysis involved comparing the behaviour of the models with that described in the SCJ Language Specification, and with running programs. This has been used to ensure that the models produced by the translation are valid, with respect to the SCJ API. Work has been done to optimise the CSP_M model to enable tractable analysis in FDR3. It also presents an interesting avenue of future work for model checking *Circus*.

In the next, and final, chapter, we evaluate the work presented in this thesis, present our conclusions, and discuss future work.

Chapter 6

Conclusion

This chapter concludes the thesis. Section 6.1 summarises the contributions that have been presented. Section 6.2 discusses the validity and utility of the presented work. Finally, Sect. 6.3 describes extensions to our approach that could be completed as future work.

6.1 Summary

Safety-Critical Java (SCJ) is a new programming language, which embeds a novel programming paradigm for safety-critical programs. To aid certification, SCJ is organised into three compliance levels, which increase in complexity from Level 0 to Level 2. Level 0 is for sequential programs that adopt a cyclic executive, where a set of computations are executed periodically. Level 1 introduces concurrency and provides both periodic and aperiodic tasks. Level 2 is the least restricted compliance level. Level 2 programs are highly concurrent, potentially multi-processor, and allow suspension and a variety of release patterns.

We aim our work at Level 2, because it has received little attention in comparison to Levels 0 and 1. For example, at the beginning of the thesis work there was no Level 2 implementation, little assessment of its features in the literature, and sparse Level 2 tool support. Even now, few tools or verification techniques are specifically aimed at Level 2.

We perform the first assessment of the features of Level 2, illuminating programming patterns for which the features are uniquely (in SCJ) useful. The assessment also provides guidelines for areas where Level 2's features could be improved to bolster it's support.

We model the programming paradigm of SCJ Level 2 as described in the SCJ language specification (v0.100). We view this paradigm as being separate to its realisation in Java. Therefore, we capture the paradigm, abstracting away from the details of its implementation in Java. The model specifies the generic behaviour that is shared by all SCJ programs. Our model is written in the state-rich process algebra *Circus*, which captures the state and behaviour of the paradigm. The model also uses elements from other languages in the *Circus* family. We use operators from *CircusTime* to capture delays and budgets, and we use *OhCircus* classes to capture non-reactive behaviour.

Finally, we provide a translation strategy that captures the application-specific behaviour of SCJ Level 2 programs and generates models representing this behaviour. These models are combined with those of the paradigm to provide a model that captures the behaviour of the whole program. We provide a formalisation of the core elements of the translation strategy, in Z. We have also developed a tool, T$^{\text{ight}}$R$^{\underline{\text{ope}}}$ to automatically generate models of SCJ programs.

6.2 Discussion

Our model of the SCJ API is the first to tackle the unique features of SCJ Level 2. It is built to have close correspondence with the SCJ API, and is based on a previous model of SCJ Level 1 [93]. However, the complexity of SCJ Level 2's unique features has meant that significant re-engineering of the model has been required. Our model also captures elements of SCJ that are not covered by the Level 1 model.

Our modelling effort has had a positive effect on SCJ, as the modelling process has illuminated some problems with its specification. Two of our suggestions for improvements have already been accepted into the language specification. The first is a simplification of the SCJ termination protocol. We used the model of the API to compare the original and simplified termination protocol, by translating the models of each from *Circus* to CSP$_M$. We found that our proposed simplified protocol had 94.5% fewer states than the original protocol. The second improvement is the rationalisation of the termination of waiting schedulables. In older versions of the language specification, waiting schedulables were not automatically woken and the only support for this was a method called only on each managed thread during termination of its controlling mission. One of our suggestions to solve this potential problem was partially adopted: the SCJ API calls the new `signalTermination()` method on each schedulable during termination. This provides a uniform way of dealing with all application-specific termination behaviour, including schedulables that may be waiting.

The framework model, which captures the SCJ API, has been constructed from the SCJ (natural) Language Specification, because it was the only description of the language that existed at the beginning of the thesis work. Subsequently, SCJ implementations have emerged, including a Level 2 implementation [46]. The fact that our model is the first of SCJ Level 2 means that it could be used as the specification for further implementations or models in other notations.

Our translation strategy has been tested by manually translating 12 example programs, which are constructed to cover the features of SCJ. They range from simple tests of SCJ's features, such as different release patterns or synchronisation and suspension, to more complex programs that use nested mission sequencers to provide concurrent missions. Further, we have developed a tool to automatically generate the *Circus* application models of a given SCJ application, called T$^{\text{ight}}$R$^{\underline{\text{ope}}}$. These techniques produce models of full programs that are valid, with respect to SCJ Language Specification.

T$^{\text{ight}}$R$^{\underline{\text{ope}}}$ provides automatic translation of valid SCJ Level 2 programs (that is programs that compile against an SCJ implementation) into *Circus* models written using our approach. The only input required from the user is the location of the SCJ program. The translation is then performed without the need for program annotations, in contrast to the tool for translating Level 1 programs [93]. However, the tool requires that each paradigm class is in a separate file. Further, complex Java statements must be rewritten: chains of method calls in one statement become several separate method calls, and if a method parameter is a new object then the object's parameters must be literal values. However, this rewriting does not affect the program's behaviour and it should be noted that these are restrictions of T$^{\text{ight}}$R$^{\text{ope}}$, not of the technique as a whole.

We have translated both the model of the API and the models of full programs to CSP$_M$ for analysis using FDR3. We have animated and model checked the CSP$_M$ version of the models. Animation was used to compare the behaviour of the models to that specified in the SCJ Language Specification and to running programs. Model checking was used to prove deadlock and divergence freedom. This technique allows program analysis and gives us confidence that our model is valid, with respect to the SCJ Language Specification.

Using *Circus* as the language in which we write our models and CSP$_M$ as the language in which we analyse them provides us with several benefits. Firstly, *Circus* is a combined notation so we get the benefits of Z, CSP, and refinement. This is useful, for example, in the *ObjectFW* process, which requires complex data operations to control waiting threads. Also, since *Circus*is written in LaTeXits models are more human-readable than CSP$_M$. Secondly, using the two different notations decouples the *Circus* model from the analysis. This is illustrated by the optimisations made to the CSP$_M$ models to allow tractable analysis. These optimisations were not incorporated into the *Circus* models, which are concerned with modelling the SCJ paradigm. It also insulates the *Circus* models from any changes in FDR3 that require the CSP$_M$ version to change.

The thesis work has shown us that *Circus* is a useful modelling language that provides adequate features for capturing the behaviour of programming languages such as SCJ. The thesis work also shows us that our modelling approach, separating the unchanging from the application-specific behaviour, is beneficial when capturing programs because it reduces the burden on translation. Further, we found it useful to have as close a correspondence between our model and SCJ as possible, as this aids traceability. For example, each of the paradigm objects in the SCJ API is represented by one process in our model – expect for the mission sequencer, which we model with two processes because of the two contexts in which mission sequencers can operate (as described in Sect. 4.2.2).

At a lower level, our modelling efforts show that care can be required when combining parallel actions that both access the same variables. The *Circus* parallel operator (introduced in Sect. 2.4) takes two name sets, each specifying the variables that one of the parallel actions can update. If two parallel actions need to update the same variable, then a third process is required to control variable access. An example of this pattern can be seen in the *MissionFW* process in Appendix. C.10, where the *Methods* action runs the *RequestTerminationMeth* action in parallel with the *TerminationPendingMeth* action, and each requires update access to the *missionTerminating* variable. To resolve this conflict, we delegate control of the variable to a third action *MissionTerminatingController*.

Overall, our work provides benefits for both the safety-critical systems and formal methods communities. Our analysis of SCJ has illuminated the utility of the unique features provided by Level 2 and helped to improve features of the language that have an impact on all three compliance levels. Our formal model of SCJ Level 2 is its first formal semantics, and enables links to a strategy that can provide refinement from abstract specifications of behaviour to concrete models of SCJ programs. Our translation, which captures SCJ Level 2 programs, validates our model and enables (via a translation to CSP) a technique for checking properties of SCJ Level 2 programs. The various threads of future work that use our contributions as a base are discussed in the next section.

6.3 Future Work

The main focus for future work is the translation. First, a full formalisation of the translation strategy in Z enables verification of the translation. This can be used to show that each function only produces a valid translation and, therefore, improve confidence in the translation. Second, improving $\mathrm{T}^{\underline{\mathrm{ight}}}\mathrm{R}^{\underline{\mathrm{ope}}}$ by removing the restrictions on the SCJ programs that it can accept will improve its applicability. Further, $\mathrm{T}^{\underline{\mathrm{ight}}}\mathrm{R}^{\underline{\mathrm{ope}}}$ is currently a terminal program, but it can be incorporated into a plugin for Eclipse. There are other popular development environments for Java, but the Icecap tool [46] (which currently provides the only implementation of SCJ Level 2) is provided as an Eclipse plugin. Such an extension for $\mathrm{T}^{\underline{\mathrm{ight}}}\mathrm{R}^{\underline{\mathrm{ope}}}$ can help to integrate it into the emerging development process for SCJ programs.

As previously mentioned, our model abstracts away from scheduling and resources. This means that it does not capture SCJ's scheduling behaviour or region-based memory management, including the global multiprocessor scheduling that is available at Level 2. These features could be added to our modelling approach to capture more concrete program information, by adding another parallel system to the *Program* process for each feature. However, both of these features require alterations to the application model and translation.

SCJ scheduling and global multiprocessor scheduling can be captured by a *Scheduling* process that stores the number of processors and the identifiers of the thread executing on each of these processors. This process should contain actions that record changes in a thread's status between running, eligible to run, blocked, or waiting. Our model already captures, as events, waiting, notification, acquiring and releasing a lock. These events can be used to synchronise with the *Scheduling* process to track a thread's status.

To implement the fixed-priority scheduler, the *ThreadFW* process must be instantiated for all threads in the program, not just those engaging in synchronisation and suspension behaviour, as in our current model. Further, *ThreadFW* needs to store the thread's affinity set (the set of processors on which it can execute), which means that the translation strategy must capture this information from the program. Control signals must be added to the framework model to allow the scheduling process to start and stop the threads within the model. The addition of another process to our model, and adding more channels, will complicate the model and possibly increase model checking times. However, it would remove the potential false negatives that our verification technique can generate, due to abstracting away from SCJ's scheduling behaviour.

SCJ's region-based memory management can be captured as a *Memory* subsystem that instantiates a *MemoryArea* process for each memory area in the SCJ program. The *Memory* subsystem should implement the abstract specification of the SCJ memory model that is described in [16]. The *MemoryArea* process must hold the names of the variables stored in the memory area, their types, and their values. Additionally, storing the nesting level of the memory area allows this information to be used for checking SCJ's memory safety rules. This could combine with extending the memory safety checking technique in [58].

In addition to the *Memory* subsystem, the application model and translation strategy must capture extra information from an SCJ program about its memory usage. For example, variable assignments and calls to `enterPrivateMemory()` must be captured. This enables the subsystem to replicate the memory behaviour of the SCJ program.

As described in Sect. 4.2, our model captures the potential for an event handler to overrun its period or deadline. Implementing checks for these potential overruns is left as future work. In our model a deadline overrun is signalled by a *release* event with the identifier of a deadline miss handler, so constructing a CSP check for this event in the model will reveal a deadline overrun. Period overrun is signalled by a *release* event occurring between the events that signal the beginning and end of the event handler's `handleAsyncEvent()` method, so checking for this will reveal a period overrun. It should be noted that this technique only works if the model terminates. Checks for other behaviours that our model captures but are undesirable can be constructed in a similar way.

The tool support for verification of SCJ Level 2 programs is lacking. Section 2.5 identifies tools for verification of Worst-Case Execution Time, Worst-Case Memory Consumption, Memory Safety, Schedulability, and Functional Correctness. A line of future work is the analysis of these verification tools to see where they fail for SCJ Level 2 programs and then extending them to cater to Level 2 programs. These extended tools and $\mathrm{T^{\underline{ight}}R^{\underline{ope}}}$ can be combined in a suite of tools that verify a range of properties of SCJ Level 2 programs.

An alternative translation approach to that in Chap. 5 is using meta-models of both SCJ and *Circus* to enable the transformation of SCJ programs into *Circus* models. This has been achieved for similar target formal languages; for example in [83], which takes two UML models (a class diagram and a state machine that describe the same system) and transforms them into a $CSP \parallel B$ model. An advantage of this approach is that the meta-models could be reused for other translations involving SCJ and *Circus*. For example, [50] provides a meta-model translation of a Domain Specific Language into a formal model, but it agnostic of the formal language used. The authors present an example translation into $CSP \parallel B$. The Z formalisation in Sect. 5.2 could be seen as a meta-model, as it defines and abstract syntax for both SCJ programs and our *Circus* model.

Model checking the *Circus* specifications of SCJ programs is a useful technique for verifying program properties, for example, deadlock and divergence freedom. Strategies exist for model checking *Circus* specifications [61, 6, 91], and these can be incorporated into $\mathrm{T^{\underline{ight}}R^{\underline{ope}}}$ to improve the workflow of model checking SCJ programs.

Our approach to animating and model checking our *Circus* models has been to translate them into CSP_M and use FDR3. Initially, this translation occasionally resulted in intractable CSP_M models due to the translation of the Z elements of *Circus* causing state explosion. Subsequently, the CSP_M models have been improved to allow tractable analysis using FDR3. However, further work is required to improve the scalability of the CSP_M model and the translation from *Circus* application models. This would allow our verification technique to cope with models of programs that are more complex.

Another line of future work is to generalise this *Circus*-to-CSP_M translation by providing a library of Z data structures captured in CSP_M in a way that is amenable to analysis in FDR.

This can provide reusable models of elements of *Circus* that allow tractable analysis. The library can be combined with the automatic translation of *Circus* models to CSP$_M$ provided by [6], which we think can produce CSP$_M$ models with the same state explosion problem as our original translation.

Adding a GUI and incorporating a *Circus*-to-CSP$_M$ translation into TightRope would provide a tool that could automatically translate SCJ Level 2 programs into *Circus*, translate the *Circus* models into CSP$_M$, and then send them directly to FDR3. The API of FDR3 allows model checking where the results are returned as a JSON string. This can be interpreted by TightRope to feedback the results of the analysis directly to the GUI.

Appendix A

Buffer Example Application

This appendix presents the buffer application, described in Sect. 2.1.2. It has one mission and two managed threads, which communicate using a one-place buffer in the mission in the familiar Readers-Writers style. This program uses the `Object.wait()` and `Object.notify()` methods, which are only available at SCJ Level 2, to control access to the buffer.

The producer schedulable suspends if the buffer is full; if not, then it writes and notifies the consumer. The consumer suspends if the buffer is empty; if not, then it removes a value from the buffer and notifies the producer. Suspension is achieved with a call to `Buffer.wait()`. Notification is achieved by a call to `Buffer.notify()`. After reading from the buffer 5 times, the consumer requests that the mission terminates. When both the managed threads have terminated, the program terminates.

A.1 BSafeletLauncher

This section presents the safelet launcher for the buffer application, which is the program entry point in the Icecap SCJ implementation. We present it here for completeness, but since it is not part of the SCJ API it is not included in out models.

```
public class BSafeletLauncher
{
  public static void main(String[] args)
  {
    new LaunchLevel2(new PCSafelet());
  }
}
```

A.2 BSafelet

This section presents the buffer application's safelet class, which controls the application.

```
public class BSafelet implements Safelet<Mission>
{
  public MissionSequencer<Mission> getSequencer()
  {
    StorageParameters storageParameters = new StorageParameters(
        Const.OVERALL_BACKING_STORE_DEFAULT - 2000000,
        Const.PRIVATE_MEM_DEFAULT, 10000 * 2, Const.MISSION_MEM_DEFAULT);

    return new MainMissionSequencer(new PriorityParameters(5), storageParameters);
  }

  @Override
  public void initializeApplication() {}

  @Override
  public long immortalMemorySize()
  {
    return Const.IMMORTAL_MEM_DEFAULT;
  }

  public Level getLevel()
  {
    return Level.LEVEL_2;
  }
}
```

A.3 MainMissionSequencer

This section presents the top-level mission sequencer of the aircraft application, which loads the main mission.

```
public class MainMissionSequencer extends MissionSequencer<Mission>
{
  private boolean returnedMission;

  public MainMissionSequencer(PriorityParameters priorityParameters,
      StorageParameters storageParameters)
  {
    super(priorityParameters, storageParameters, null);
    returnedMission = false;
  }

  protected Mission getNextMission()
  {
    if (!returnedMission)
    {
      returnedMission = true;
      return new MainMission();
    }
    else
    {
      return null;
    }
  }
}
```

A.4 MainMission

This section presents the main mission of the buffer application, which holds the buffer object and registers two schedulables.

```java
public class MainMission extends Mission
{
  private final Buffer buffer;

  public MainMission()
  {
    Services.setCeiling(this, 20);
    buffer = new Buffer();
  }

  protected void initialize()
  {
    StorageParameters storageParameters = new StorageParameters(150 * 1000,
        Const.PRIVATE_MEM_DEFAULT, Const.IMMORTAL_MEM_DEFAULT,
        Const.MISSION_MEM_DEFAULT - 100 * 1000);

    new Producer(new PriorityParameters(10), storageParameters, this).register();
    new Consumer(new PriorityParameters(10), storageParameters, this).register();
  }

  public Buffer getBuffer()
  {
    return buffer;
  }

  public boolean cleanUp()
  {
    return false;
  }

  public long missionMemorySize()
  {
    return 1048576;
  }
}
```

A.5 Buffer

This section presents the `Buffer` class, which controls access to a one-place buffer using suspension.

```
public class Buffer
{
  private volatile int theBuffer;

  public Buffer()
  {
    theBuffer = 0;
    Services.setCeiling(this, 20);
  }

  public boolean bufferEmpty(String name)
  {
    return theBuffer == 0;
  }

  public synchronized void write(int update) throws InterruptedException
  {
    while (!bufferEmpty("Producer"))
    {
      this.wait();
    }

    theBuffer = update;
    this.notify();
  }

  public synchronized int read() throws InterruptedException
  {
    while (bufferEmpty("Consumer"))
    {
      this.wait();
    }

    int out = theBuffer;
    theBuffer = 0;
    this.notify();

    return out;
  }
}
```

A.6 Producer

This section presents the `Producer` managed thread, which writes data to the shared buffer.

```java
public class Producer extends ManagedThread
{
  private final MainMission mainMission;
  private final Buffer buffer;

  public Producer(PriorityParameters priority, StorageParameters storage,
      MainMission mainMission)
  {
    super(priority, storage, null);

    this.mainMission = mainMission;
    buffer = mainMission.getBuffer();
  }

  public void run()
  {
    int i = 1;

    while (!mainMission.terminationPending())
    {
      try
      {
        buffer.write(i);
      }
      catch (InterruptedException e)
      {
        e.printStackTrace();
      }

      i++;

      if ( i >= 5)
      {
        pcMission.requestTermination();
      }
    }
  }
}
```

A.7 Consumer

This section presents the `Consumer` managed thread, which reads data from the shared buffer.

```java
public class Consumer extends ManagedThread
{
  private final MainMission mainMission;
  private final Buffer buffer;

  public Consumer(PriorityParameters priority, StorageParameters storage,
      MainMission mainMission)
  {
    super(priority, storage, null);

    this.mainMission = mainMission;
    buffer = mainMission.getBuffer();
  }

  public void run()
  {
    while (!mainMission.terminationPending())
    {
      int result = 999;
      try
      {
        result = buffer.read();
      }
      catch (InterruptedException e)
      {
        e.printStackTrace();
      }
    }
  }
}
```

Appendix B

Aircraft Example Application

This appendix presents the aircraft example application, described in Sect. 2.1.3, which controls a simplified aircraft. It has several persistent schedulable objects, which represent things about the aircraft that always need monitoring or handling; and three missions: `TakeOff`, `Cruise`, and `Land`, that represent the aircraft's phases of flight. These mission representing modes of operation are controlled by the mode changer, which is a mission sequencer. As this is an abstract example application these persistent handlers simplify what in reality would be more complex systems. Each of these schedulable objects could be implemented as multiple schedulable objects, possibly in their own nested mission – for structuring purposes.

Upon termination of the `Land` mission, or in the event of a failure causing the termination of the `TakeOff` mission, the application terminates. As this is an abstract example any remedial actions taken in the event of failures or any cleanup actions at the end of a successful landing are omitted.

B.1 ACSafeletLauncher

This section presents the safelet launcher for the aircraft application, which is the program entry point in the Icecap SCJ implementation. We present it here for completeness, but since it is not part of the SCJ API it is not included in out models.

```
public class ACSafeletLauncher
{
  public static void main(String[] args)
  {
    ACSafelet GERTI = new ACSafelet();
    new LaunchLevel2(GERTI);
  }
}
```

B.2 ACSafelet

This section presents the safelet that controls the aircraft application.

```java
public class ACSafelet implements Safelet<Mission>
{
  @Override
  public MissionSequencer<Mission> getSequencer()
  {
    StorageParameters storageParameters = new StorageParameters(150 * 1000,
        Const.PRIVATE_MEM_DEFAULT - 25 * 1000,
        Const.IMMORTAL_MEM_DEFAULT - 50 * 1000,
        Const.MISSION_MEM_DEFAULT - 100 * 1000);

    return new MainMissionSequencer(new PriorityParameters(5),
        storageParameters);
  }

  @Override
  public void initializeApplication() {}

  @Override
  public long immortalMemorySize()
  {
    return Const.IMMORTAL_MEM_DEFAULT;
  }
}
```

B.3 MainMissionSequencer

This section presents the aircraft top-level mission sequencer, which loads the main mission.

```java
public class MainMissionSequencer extends MissionSequencer<Mission>
{
  private boolean returnedMission;

  public MainMissionSequencer(PriorityParameters priority,
      StorageParameters storage)
  {
    super(priority, storage, null);
    returnedMission = false;
  }

  @Override
  protected Mission getNextMission()
  {
    if (!returnedMission)
    {
      returnedMission = true;
      return new MainMission();
    } else
    {
      return null;
    }
  }
}
```

B.4 MainMission

This section presents the `MainMission` class, which registers the aircraft's four persistent schedulables and the nested mission sequencer that controls the three modes that the aircraft's software can be in.

```java
public class MainMission extends Mission
{
  private double cabinPressure;
  private double emergencyOxygen;
  private double fuelRemaining;

  private double altitude;
  private double airSpeed;
  private double heading;

  @Override
  protected void initialize()
  {
    StorageParameters storageParameters = new StorageParameters(150 * 1000,
        Const.PRIVATE_MEM_DEFAULT - 25 * 1000,
        Const.IMMORTAL_MEM_DEFAULT - 50 * 1000,
        Const.MISSION_MEM_DEFAULT - 100 * 1000);

    StorageParameters storageParametersSchedulable = new StorageParameters(
        Const.PRIVATE_MEM_DEFAULT - 30 * 1000,
        Const.PRIVATE_MEM_DEFAULT - 30 * 1000,
        Const.IMMORTAL_MEM_DEFAULT - 50 * 1000,
        Const.MISSION_MEM_DEFAULT - 100 * 1000);

    ACModeChanger2 aCModeChanger = new ACModeChanger2(new PriorityParameters(
        5), storageParameters, this);

    aCModeChanger.register();

    EnvironmentMonitor environmentMonitor = new EnvironmentMonitor(
        new PriorityParameters(5), new PeriodicParameters(
            new RelativeTime(10, 0), null),
        storageParametersSchedulable, "Environment Monitor", this);

    environmentMonitor.register();

    ControlHandler controlHandler = new ControlHandler(
        new PriorityParameters(5), new AperiodicParameters(new RelativeTime(10, 0),
            null),
        storageParametersSchedulable, "Control Handler");

    controlHandler.register();

    FlightSensorsMonitor flightSensMon = new FlightSensorsMonitor(
        new PriorityParameters(5), new PeriodicParameters(
            new RelativeTime(10, 0), null),
        storageParametersSchedulable, "Flight Sensors Monitor", this);

    flightSensMon.register();

    CommunicationsHandler commsHandler = new CommunicationsHandler(
        new PriorityParameters(5), new AperiodicParameters(),
        storageParametersSchedulable, "Communications Handler");

    commsHandler.register();

  }

  @Override
  public long missionMemorySize()
  {
    return Const.MISSION_MEM_DEFAULT;
  }

  public double getAirSpeed()
  {
    return airSpeed;
  }

  public double getAltitude()
  {
    return altitude;
  }

  public double getCabinPressure()
  {
    return cabinPressure;
  }

  public double getEmergencyOxygen()
  {
    return emergencyOxygen;
  }

  public double getFuelRemaining()
  {
    return fuelRemaining;
  }

  public double getHeading()
  {
    return heading;
  }

  public void setAirSpeed(double newAirSpeed)
  {
    this.airSpeed = newAirSpeed;
  }

  public void setAltitude(double newAltitude)
  {
    this.altitude = newAltitude;
  }

  public void setCabinPressure(double newCabinPressure)
  {
    this.cabinPressure = newCabinPressure;
  }

  public void setEmergencyOxygen(double newEmergencyOxygen)
  {
    this.emergencyOxygen = newEmergencyOxygen;
  }

  public void setFuelRemaining(double newFuelRemaining)
  {
    this.fuelRemaining = newFuelRemaining;
  }

  public void setHeading(double newHeading)
  {
    this.heading = newHeading;
  }
}
```

B.5 ControlHandler

This section present the `ControlHandler`, a persistent schedulable that handles the aircraft's controls.

```
public class ControlHandler extends AperiodicEventHandler
{
  public ControlHandler(PriorityParameters priority,
      AperiodicParameters release, StorageParameters storage, String name)
  {
    super(priority, release, storage, null);
  }

  @Override
  public void handleAsyncEvent()
  {
    //Handle Control Signals
  }
}
```

B.6 FlightSensorsMonitor

This section present the `FlightSensorsMonitor`, a persistent schedulable that monitors the aircraft's flight sensors.

```
public class FlightSensorsMonitor extends PeriodicEventHandler
{
  private MainMission controllingMission;

  public FlightSensorsMonitor(PriorityParameters priority,
      PeriodicParameters periodic, StorageParameters storage,
      String name, MainMission controllingMission)
  {
    super(priority, periodic, storage, null);
    this.controllingMission = controllingMission;
  }

  @Override
  public void handleAsyncEvent()
  {
    // read air speed
    controllingMission.setAirSpeed(0);
    // read altitude
    controllingMission.setAltitude(0);
    // read heading
    controllingMission.setHeading(0);
  }
}
```

B.7 EnvironmentMonitor

This section present the `EnvironmentMonitor`, a persistent schedulable that monitors the aircraft's environment.

```java
public class EnvironmentMonitor extends PeriodicEventHandler
{
  private MainMission controllingMission;

  public EnvironmentMonitor(PriorityParameters priority,
      PeriodicParameters periodic,
      StorageParameters storage,
      String name,
      MainMission controllingMission)
  {
    super(priority, periodic, storage, null);
    this.controllingMission = controllingMission;
  }

  @Override
  public void handleAsyncEvent()
  {
    // read cabin pressure from sensors
    controllingMission.setCabinPressure(0);

    // read emergency Oxygen Levels
    controllingMission.setEmergencyOxygen(0);

    // read remaining fuel
    controllingMission.setFuelRemaining(0);
  }
}
```

B.8 CommunicationsHandler

This section present the `CommunicationsHandler`, a persistent schedulable that monitors the aircraft's communication systems.

```java
public class CommunicationsHandler extends AperiodicEventHandler
{
  public CommunicationsHandler(PriorityParameters priority,
      AperiodicParameters release, StorageParameters storage, String name)
  {
    super(priority, release, storage, null);
  }

  @Override
  public void handleAsyncEvent()
  {
    //Handle Communication Signal
  }
}
```

B.9 ACModeChanger

This section presents the nested mission sequencer that controls the three modes of the aircraft system: taking off, cruising, and landing. Each mode is represented by a mission that is loaded by this mission sequencer.

```
public class ACModeChanger extends MissionSequencer<Mission>
{
  private MainMission controllingMission;

  public ACModeChanger(PriorityParameters priority,
      StorageParameters storage, MainMission controllingMission)
  {
    super(priority, storage, null);
    this.controllingMission = controllingMission;
  }

  private int modesLeft = 3;

  public ACModeChanger(PriorityParameters priority, StorageParameters storage)
  {
    super(priority, storage, null);
  }

  @Override
  protected Mission getNextMission()
  {
    if (modesLeft == 3)
    {
      modesLeft--;
      return new TakeOffMission(controllingMission);
    } else if (modesLeft == 2)
    {
      modesLeft--;
      return new CruiseMission(controllingMission);
    } else if (modesLeft == 1)
    {
      modesLeft--;
      return new LandMission(controllingMission);
    } else
    {
      return null;
    }
  }
}
```

B.10 TakeOffMission

This section presents the `TakeOffMission`, which registers the schedulables that are specific to the take off mode. It implements the `LandingGearUser` interface because it uses the landing gear.

```
public class TakeOffMission extends Mission
{
  private final double SAFE_AIRSPEED_THRESHOLD = 10.00;
  private final double TAKEOFF_ALTITUDE = 10.00;
  private MainMission controllingMission;
  private boolean abort = false;
  private boolean landingGearDeployed;

  public TakeOffMission(MainMission controllingMission)
  {
    this.controllingMission = controllingMission;
  }

  @Override
  protected void initialize()
  {
    StorageParameters storageParametersSchedulable = new StorageParameters(
        Const.PRIVATE_MEM_DEFAULT - 30 * 1000,
        Const.PRIVATE_MEM_DEFAULT - 30 * 1000,
        Const.IMMORTAL_MEM_DEFAULT - 50 * 1000,
        Const.MISSION_MEM_DEFAULT - 100 * 1000);

    LandingGearHandler landingGearHandler = new LandingGearHandler(
        new PriorityParameters(5), new AperiodicParameters(),
        storageParametersSchedulable, "Landing Gear Handler", this);

    landingGearHandler.register();

    TakeOffMonitor takeOffMonitor = new TakeOffMonitor(
        new PriorityParameters(5), new PeriodicParameters(
            new RelativeTime(0, 0), new RelativeTime(500, 0)),
        storageParametersSchedulable, controllingMission, this, TAKEOFF_ALTITUDE,
        landingGearHandler);

    takeOffMonitor.register();

    TakeOffFailureHandler takeOffFailureHandler = new TakeOffFailureHandler(
        new PriorityParameters(5), new AperiodicParameters(),
        storageParametersSchedulable, "Take Off Handler", controllingMission, this,
        SAFE_AIRSPEED_THRESHOLD);

    takeOffFailureHandler.register();
  }

  @Override
  public long missionMemorySize()
  {
    return Const.MISSION_MEM_DEFAULT;
  }

  @Override
  public boolean cleanUp()
  {
    return !abort;
  }

  public void takeOffAbort()
  {
    abort = true;
  }

  public void deployLandingGear()
  {
    landingGearDeployed = true;
  }

  public void stowLandingGear()
  {
    landingGearDeployed = false;
  }

  public boolean isLandingGearDeployed()
  {
    return landingGearDeployed;
  }
}
```

B.11 LandingGearHandlerTakeOff

This section presents the `LandingGearHandlerTakeOff`, which stows the aircraft's landing gear when it reaches a certain altitude.

```java
public class LandingGearHandlerTakeOff extends AperiodicEventHandler
{
  private final TakeOffMission mission;

  public LandingGearHandler(PriorityParameters priority,
      AperiodicParameters release, StorageParameters storage,
      String name, TakeOffMission mission)
  {
    super(priority, release, storage, null);
    this.mission = mission;
  }

  @Override
  public void handleAsyncEvent()
  {
    boolean landingGearIsDeployed = mission.isLandingGearDeployed();

    if (landingGearIsDeployed)
    {
      mission.stowLandingGear();
    } else
    {
      mission.deployLandingGear();
    }
  }
}
```

B.12 TakeOffMonitor

This section presents the `TakeOffMonitor`, which monitors the aircraft's altitude during take off and triggers the termination of the `TakeOffMission` when a certain altitude is reached.

```
public class TakeOffMonitor extends PeriodicEventHandler
{
  private final MainMission mainMission ;
  private final TakeOffMission takeOffMission;
  private double takeOffAltitude;
  private AperiodicEventHandler landingGearHandler;

  public TakeOffMonitor(PriorityParameters priority,
      PeriodicParameters periodic, StorageParameters storage,
      MainMission mainMission, TakeOffMission takeOffMission, double takeOffAltitude,
      AperiodicEventHandler landingGearHandler)
  {
    super(priority, periodic, storage, null);
    this.mainMission = mainMission;
    this.takeOffMission = takeOffMission;
    this.takeOffAltitude = takeOffAltitude;
    this.landingGearHandler = landingGearHandler;

  }

  @Override
  public void handleAsyncEvent()
  {
    double altitude = mainMission.getAltitude();

    if (altitude > takeOffAltitude)
    {
      landingGearHandler.release();
      takeOffMission.requestTermination();
    }
  }
}
```

B.13 TakeOffFailureHandler

This section presents the TakeOffFailureHandler, which aborts the take off and terminates the main mission (and, therefore, the application) if the correct take off speed is not reached when the handler is released.

```java
public class TakeOffFailureHandler extends AperiodicEventHandler
{
  private final MainMission mainMission;
  private final TakeOffMission takeoffMission;
  private double threshold;

  public TakeOffFailureHandler(PriorityParameters priority,
      AperiodicParameters release, StorageParameters storage,
      String name, MainMission mainMission, TakeOffMission takeoffMission, Double
          threshold)
  {
    super(priority, release, storage, null);
    this.takeoffMission = takeoffMission;
    this.mainMission = mainMission;
    this.threshold = threshold;
  }

  @Override
  public void handleAsyncEvent()
  {
    double currentSpeed = mainMission.getAirSpeed();

    if (currentSpeed < threshold)
    {
      // Failure Abort
      takeoffMission.takeOffAbort();
      takeoffMission.requestTermination();
    } else
    {
      // Failure: Continue and Land
    }
  }

}
```

B.14 CruiseMission

This section presents `CruiseMission`, which registers the schedulables that are specific to the cruise mode.

```
public class CruiseMission extends Mission
{
  private final MainMission controllingMission;

  public CruiseMission(MainMission controllingMission)
  {
    this.controllingMission = controllingMission;
  }

  @Override
  protected void initialize()
  {
    StorageParameters storageParametersSchedulable = new StorageParameters(
        Const.PRIVATE_MEM_DEFAULT - 30 * 1000,
        Const.PRIVATE_MEM_DEFAULT - 30 * 1000,
        Const.IMMORTAL_MEM_DEFAULT - 50 * 1000,
        Const.MISSION_MEM_DEFAULT - 100 * 1000);

    BeginLandingHandler beginLandingHandler = new BeginLandingHandler(
        new PriorityParameters(5), new AperiodicParameters(),
        storageParametersSchedulable, "Begin Landing Handler", controllingMission);
    beginLandingHandler.register();

    int maxP = PriorityScheduler.instance().getMaxPriority();

    NavigationMonitor navigationMonitor = new NavigationMonitor(
        new PriorityParameters(5), new PeriodicParameters(
            new RelativeTime(0, 0), new RelativeTime(10, 0)),
        storageParametersSchedulable, "Cruise Controller", controllingMission);
    navigationMonitor.register();
  }

  @Override
  public long missionMemorySize()
  {
    return Const.MISSION_MEM_DEFAULT;
  }
}
```

B.15 BeginLandingHandler

This section presents the `BeginLandingHandler`, which triggers the termination of the mission `CruiseMission`, so that the landing mode can begin.

```java
public class BeginLandingHandler extends AperiodicEventHandler
{
  private Mission controllingMission;

  public BeginLandingHandler(PriorityParameters priority,
      AperiodicParameters release, StorageParameters storage,
      String name, Mission controllingMission)
  {
    super(priority, release, storage, null);
    this.controllingMission = controllingMission;
  }

  @Override
  public void handleAsyncEvent()
  {
    controllingMission.requestTermination();
  }
}
```

B.16 NavigationMonitor

This section presents the `NavigationMonitor`, which provides the navigation services required during normal flight.

```java
public class NavigationMonitor extends PeriodicEventHandler
{
  private final MainMission mainMission;

  public NavigationMonitor(PriorityParameters priority,
      PeriodicParameters periodic, StorageParameters storage,
      String name, MainMission mainMission)
  {
    super(priority, periodic, storage, null);
    this.mainMission = mainMission;
  }

  @Override
  public void handleAsyncEvent()
  {
    // Read and check these variables
    double heading = mainMission.getHeading();
    double airSpeed = mainMission.getAirSpeed();
    double altitude = mainMission.getAltitude();

    // Check the variables again expected values
  }
}
```

B.17 LandMission

This section presents the `LandMission`, which registers the schedulables that are specific to the land mode. It implements the `LandingGearUser` interface because it uses the landing gear.

```
public class LandMission extends Mission
{
  private final MainMission controllingMission;

  final double SAFE_LANDING_ALTITUDE = 10.00;
  final double ALTITUDE_READING_ON_GROUND = 0.0;

  private boolean abort = false;

  public LandMission(MainMission controllingMission)
  {
    this.controllingMission = controllingMission;
  }

  private boolean landingGearDeployed;

  @Override
  protected void initialize()
  {

    StorageParameters storageParametersSchedulable = new StorageParameters(
        Const.PRIVATE_MEM_DEFAULT - 30 * 1000,
        Const.PRIVATE_MEM_DEFAULT - 30 * 1000,
        Const.IMMORTAL_MEM_DEFAULT - 50 * 1000,
        Const.MISSION_MEM_DEFAULT - 100 * 1000);

    GroundDistanceMonitor groundDistanceMonitor = new GroundDistanceMonitor(
        new PriorityParameters(5), new PeriodicParameters(
            new RelativeTime(0, 0), new RelativeTime(10, 0)),
        storageParametersSchedulable, controllingMission, ALTITUDE_READING_ON_GROUND);
    groundDistanceMonitor.register();

    LandingGearHandlerLand landingHandler = new LandingGearHandlerLand(
        new PriorityParameters(5), new AperiodicParameters(),
        storageParametersSchedulable, "Landing Handler", this);

    landingHandler.register();

    InstrumentLandingSystemMonitor ilsMonitor = new InstrumentLandingSystemMonitor(
        new PriorityParameters(5), new PeriodicParameters(
            new RelativeTime(0, 0), new RelativeTime(10, 0)),
        storageParametersSchedulable, "ILS Monitor", this);
    ilsMonitor.register();

    SafeLandingHandler safeLandingHandler = new SafeLandingHandler(
        new PriorityParameters(5), new AperiodicParameters(),
        storageParametersSchedulable, "Safe Landing Handler", controllingMission,
        SAFE_LANDING_ALTITUDE);

    safeLandingHandler.register();
  }

  @Override
  public long missionMemorySize()
  {
    return Const.MISSION_MEM_DEFAULT;
  }

  public void deployLandingGear()
  {
    landingGearDeployed = true;
  }

  public void stowLandingGear()
  {
    landingGearDeployed = false;
  }

  public boolean isLandingGearDeployed()
  {
    return landingGearDeployed;
  }

  @Override
  public boolean cleanUp()
  {
    return false;
  }
}
```

B.18 GroundDistanceMonitor

This section presents the `GroundDistanceMonitor`, which monitors the aircraft's distance from the ground and terminates the `MainMission` when the aircraft has landed.

```java
public class GroundDistanceMonitor extends PeriodicEventHandler
{
  private final MainMission mainMission;
  private final double readingOnGround;

  public GroundDistanceMonitor(PriorityParameters priority,
      PeriodicParameters periodic, StorageParameters storage,
      MainMission mainMission, double readingOnGround)
  {
    super(priority, periodic, storage, null);

    this.mainMission = mainMission;
    this.readingOnGround = readingOnGround;
  }

  @Override
  public void handleAsyncEvent()
  {
    // Read this value from sensors
    double distance = mainMission.getAltitude();

    if (distance == readingOnGround)
    {
      mainMission.requestTermination();
    }
  }
}
```

B.19 LandingGearHandlerLand

This section presents the `LandingGearHandlerLand`, which deploys the aircraft's landing gear at a certain altitude.

```
public class LandingGearHandlerLand extends AperiodicEventHandler
{
  private final LandMission mission;

 public LandingGearHandlerLand(PriorityParameters priority,
      AperiodicParameters release, StorageParameters storage,
      String name, LandMission mission)
  {
    super(priority, release, storage, null);
    this.mission = mission;
  }

  @Override
  public void handleAsyncEvent()
  {
    boolean landingGearIsDeployed = mission.isLandingGearDeployed();

    if (landingGearIsDeployed)
    {
      mission.stowLandingGear();
    } else
    {
      mission.deployLandingGear();
    }
  }
}
```

Appendix C

Framework Model

This appendix presents our entire model of the SCJ API (the framework model) written in *Circus*. It captures the unchanging behaviour of SCJ Level 2 programs.

C.1 GlobalTypes

section *GlobalTypes* **parents** *scj_prelude*, *SchedulableId*

$[ThreadID]$
$[ObjectID]$
$[NonParadigmID]$
$[totalThreads]$

$$
\begin{array}{|l}
SafeletTId : ThreadID \\
nullThreadId : ThreadID
\end{array}
$$

$ThreadMap == ThreadID \pfun \mathbb{N}_1$

$ExceptionType ::= interruptedException \mid illegalMonitorStateException \mid$
$\quad illegalArgumentException \mid illegalThreadStateException \mid$
$\quad illegalStateException \mid ceilingViolationException$

$maxNanos == 999999$

$AperiodicType ::= aperiodic \mid aperiodicLong$

$$
\begin{array}{|l}
\mathbb{R} : \mathbb{P}\,\mathbb{A} \\
\hline
\mathbb{Z} \subset \mathbb{R}
\end{array}
$$

C.2 Priority

section *Priority* **parents** *scj_prelude*

$$\begin{array}{|l}
MinPriority : \mathbb{N}_1 \\
MaxPriority : \mathbb{N}_1 \\
\hline
MaxPriority - MinPriority \geq 2
\end{array}$$

$PriorityLevel == MinPriority \mathrel{..} MaxPriority$

C.3 Priority Queue

section *PriorityQueue* **parents** *scj_prelude*, *GlobalTypes*, *Priority*

$$\begin{array}{|l}
PriorityQueue == PriorityLevel \to (\operatorname{iseq} ThreadID) \\
\hline
\forall pq : PriorityQueue \bullet nullThreadId \notin \operatorname{ran}(\bigcup(\operatorname{ran} pq))
\end{array}$$

$$\begin{array}{|l}
IsEmpty : PriorityQueue \to \mathbb{B} \\
\hline
\forall pq : PriorityQueue \mid (\bigcup(pq \llparenthesis PriorityLevel \rrparenthesis)) = \varnothing \bullet \\
\quad IsEmpty(pq) = \mathbf{True}
\end{array}$$

$$\begin{array}{|l}
AddToPriorityQueue : PriorityQueue \times ThreadID \times PriorityLevel \to PriorityQueue \\
\hline
\forall pq : PriorityQueue;\ t : ThreadID;\ p : PriorityLevel \mid \\
\quad t \neq nullThreadId \land \\
\quad t \notin \operatorname{ran}(\bigcup(\operatorname{ran}(pq))) \bullet \\
AddToPriorityQueue(pq, t, p) = (pq \oplus \{p \mapsto pq(p) \frown \langle t \rangle\})
\end{array}$$

$$\begin{array}{|l}
RemoveFromPriorityQueue : PriorityQueue \rightarrowtail PriorityQueue \times ThreadID \\
\hline
(\forall pq : PriorityQueue \bullet \\
\quad (\exists t : ThreadID;\ p : PriorityLevel \mid \\
\qquad p = max \{pl : PriorityLevel \mid pq(pl) \neq \langle\rangle\} \land \\
\qquad t = head\ pq(p) \\
\qquad \bullet RemoveFromPriorityQueue(pq) = (pq \oplus \{p \mapsto tail\ pq(p)\}, t)))
\end{array}$$

$$\begin{array}{|l}
RemoveThreadFromPriorityQueue : PriorityQueue \times ThreadID \times PriorityLevel \\
\quad \to PriorityQueue \\
\hline
\forall pq : PriorityQueue;\ t : ThreadID;\ p : PriorityLevel \mid \\
\quad pq(p) \upharpoonright \{t\} \neq \langle\rangle \bullet \\
\quad RemoveThreadFromPriorityQueue(pq, t, p) = pq \oplus \{p \mapsto squash\,(pq(p) \rhd \{t\})\}
\end{array}$$

$$
\begin{array}{|l}
ElementsOf : PriorityQueue \rightarrow \mathbb{P}\, ThreadID \\
\hline
\forall pq : PriorityQueue \mid pq \neq \varnothing \bullet \\
\quad (\exists elems : \mathbb{P}\, ThreadID \mid \\
\qquad elems = \bigcup(\mathrm{ran} \llparenthesis \mathrm{ran}\, pq \rrparenthesis) \\
\qquad \bullet ElementsOf(pq) = elems)
\end{array}
$$

C.4 Ids

C.4.1 MissionId

section *MissionId*

$[MissionID]$

$$
\begin{array}{|l}
nullMissionId : MissionID
\end{array}
$$

C.4.2 SchedulableId

section *SchedulableId*

$[SchedulableID]$

$$
\begin{array}{|l}
TopLevelSequencerId : SchedulableID \\
nullSequencerId : SchedulableID \\
nullSchedulableId : SchedulableID
\end{array}
$$

C.5 Channels

C.5.1 FrameworkChan

section *FrameworkChan* **parents** *GlobalTypes*

channel $\mathit{throw} : \mathit{ExceptionType}$
channel *done_toplevel_sequencer*

C.5.2 ServicesChan

section *ServicesChan* **parents** *scj_prelude*, *GlobalTypes*, *MissionId*, *Priority*, *PriorityQueue*

channel $\mathit{setCeilingPriority} : \mathit{MissionID} \times \mathit{ObjectID} \times \mathit{PriorityLevel}$

C.5.3 ObjectChan

section *ObjectChan* **parents** *ObjectFWChan*, *ObjectMethChan*, *ServicesChan*

channelset *MonitorSync* == {| *fully_unlock*, *relock*, *relock_this* |}
channelset *MLCSync* == {| *relock_this*, *lock_request*, *lockAcquired*, *get_lockedBy*,
reset_lockedBy, *fully_unlock* |}
channelset *CPCSync* == {| *setCeilingPriority*, *get_ceilingPriority* |}
channelset *WaitSync* == {| *cancel_wait_timer*, *waitRet*, *waitForObjectRet* |}
channelset *WQSync* == {| *add_to_wait*, *remove_from_wait*, *remove_most_eligible_from_wait* |}
channelset *InterruptSync* == {| *remove_from_wait*, *get_waitQueue* |}

C.5.4 ObjectFWChan

section *ObjectFWChan* **parents** *GlobalTypes*, *Priority*, *PriorityQueue*, *JTime*

$\mathit{WaitType} ::= \mathit{wait} \mid \mathit{waitForObject}$

channel $\mathit{unlock\_Monitor} : \mathit{ObjectID} \times \mathit{ThreadID}$

channel $\mathit{relock} : \mathit{ObjectID} \times \mathit{ThreadID}$

channel $\mathit{relock\_this} : \mathit{ObjectID} \times \mathit{ThreadID}$

channel $\mathit{startSyncMeth} : \mathit{ObjectID} \times \mathit{ThreadID}$

channel $\mathit{lockAcquired} : \mathit{ObjectID} \times \mathit{ThreadID}$

channel $\mathit{endSyncMeth} : \mathit{ObjectID} \times \mathit{ThreadID}$

channel $\mathit{cancel\_wait\_timer} : \mathit{ObjectID} \times \mathit{ThreadID}$

channel $\mathit{start\_timer} : \mathit{ObjectID} \times \mathit{ThreadID} \times \mathit{PriorityLevel} \times \mathit{JTime}$

channel $\mathit{lock\_request} : \mathit{ObjectID} \times \mathit{ThreadID}$

channel $\mathit{assignLock} : \mathit{ObjectID}$

channel $\mathit{add\_to\_wait} : \mathit{ObjectID} \times \mathit{ThreadID} \times \mathit{PriorityLevel} \times \mathit{WaitType}$

channel $\mathit{remove\_from\_wait} : \mathit{ObjectID} \times \mathit{ThreadID} \times \mathit{PriorityLevel}$

channel $\mathit{remove\_most\_eligible\_from\_wait} : \mathit{ObjectID}$

channel $\mathit{removed\_thread} : \mathit{ObjectID} \times \mathit{ThreadID} \times \mathit{WaitType}$

channel $\mathit{get\_lockedBy} : \mathit{ObjectID} \times \mathit{ThreadID}$

channel $\mathit{get\_waitQueue} : \mathit{ObjectID} \times \mathit{PriorityQueue}$

channel $\mathit{reset\_lockedBy} : \mathit{ObjectID}$

channel $\mathit{fully\_unlock} : \mathit{ObjectID}$

channel $\mathit{increment\_locks} : \mathit{ObjectID}$

channel $\mathit{decrement\_locks} : \mathit{ObjectID} \times \mathbb{Z}$

channel $\mathit{get\_ceilingPriority} : \mathit{ObjectID} \times \mathit{PriorityLevel}$

channel $\mathit{start\_waitForObject\_timer} : \mathit{ObjectID} \times \mathit{ThreadID} \times \mathit{PriorityLevel} \times \mathit{JTime}$

channel $\mathit{get\_waitForObjectThreads} : \mathit{ObjectID} \times \mathbb{P}\ \mathit{ThreadID}$

C.5.5 ObjectMethChan

section *ObjectMethChan* **parents** *GlobalTypes*, *Priority*, *JTime*

channel $\mathit{waitCall} : \mathit{ObjectID} \times \mathit{ThreadID}$

channel $\mathit{timedWaitCall} : \mathit{ObjectID} \times \mathit{ThreadID} \times \mathit{JTime}$

channel $\mathit{waitRet} : \mathit{ObjectID} \times \mathit{ThreadID}$

channel $\mathit{waitForObjectCall} : \mathit{ObjectID} \times \mathit{ThreadID} \times \mathit{JTime}$

channel $\mathit{waitForObjectRet} : \mathit{ObjectID} \times \mathit{ThreadID} \times \mathbb{B}$

channel $\mathit{notify} : \mathit{ObjectID} \times \mathit{ThreadID}$

channel $\mathit{notifyAll} : \mathit{ObjectID} \times \mathit{ThreadID}$

C.5.6 ThreadChan

section *ThreadChan* **parents** *ThreadFWChan*, *ThreadMethChan*

C.5.7 ThreadFWChan

section *ThreadFWChan* **parents** *GlobalTypes*, *Priority*

channel $get\_priorityLevel : ThreadID \times ObjectID \times PriorityLevel$
channel $raise\_thread\_priority : ThreadID \times PriorityLevel$
channel $lower\_thread\_priority : ThreadID$
channel $set\_interrupted : ThreadID \times \mathbb{B}$
channel $get\_interrupted : ThreadID \times \mathbb{B}$

C.5.8 ThreadMethChan

section *ThreadMethChan* **parents** *GlobalTypes*

channel $interrupt : ThreadID$
channel $isInterruptedCall : ThreadID$
channel $isInterruptedRet : ThreadID \times \mathbb{B}$
channel $interruptedCall : ThreadID$
channel $interruptedRet : ThreadID \times \mathbb{B}$

C.5.9 SafeletChan

section *SafeletChan* **parents** *SafeletFWChan*, *SafeletMethChan*

channelset $SafeletAppSync == \{\!| getSequencerCall,$
$getSequencerRet, initializeApplicationCall, initializeApplicationRet,$
$end\_safelet\_app |\!\}$

C.5.10 SafeletFWChan

section *SafeletFWChan* **parents** *scj_prelude*

channel $end\_safelet\_app$
channel $done\_safeletFW$

C.5.11 SafeletMethChan

section *SafeletMethChan* **parents** *scj_prelude*, *SchedulableId*, *MissionId*

channel $initializeApplicationCall$
channel $initializeApplicationRet$
channel $getSequencerCall$
channel $getSequencerRet : SchedulableID$
channel $checkSchedulable : MissionID \times \mathbb{B}$
channel $deregister : \mathbb{F}\ SchedulableID$

C.5.12 MissionSequencerChan

section $MissionSequencerChan$ **parents** $scj\_prelude$, $MissionId$, $SchedulableId$, $MissionSequencerMethChan$, $MissionSequencerFWChan$

channelset $MissionSequencerAppSync == \{\!| getNextMissionCall, getNextMissionRet, end\_sequencer\_app |\!\}$

C.5.13 MissionSequencerFWChan

section $MissionSequencerFWChan$ **parents** $scj\_prelude$, $MissionId$, $SchedulableId$

channel $get\_continue : SchedulableID \times \mathbb{B}$
channel $end\_sequencer\_app : SchedulableID$
channel $end\_methods : SchedulableID$
channel $end\_terminations : SchedulableID$

C.5.14 MissionSequencerMethChan

section $MissionSequencerMethChan$ **parents** $scj\_prelude$, $MissionId$, $SchedulableId$

channel $getNextMissionCall : SchedulableID$
channel $getNextMissionRet : (SchedulableID \times MissionID)$
channel $requestSequenceTermination : (SchedulableID \times \mathbb{B})$
channel $sequenceTerminationPendingCall : SchedulableID$
channel $sequenceTerminationPendingRet : (SchedulableID \times \mathbb{B})$

C.5.15 TopLevelMissionSequencerChan

section $TopLevelMissionSequencerChan$ **parents** $TopLevelMissionSequencerFWChan$, $MissionSequencerChan$

C.5.16 TopLevelMissionSequencerFWChan

section $\mathit{TopLevelMissionSequencerFWChan}$ **parents** $\mathit{scj\_prelude}, \mathit{MissionSequencerFWChan},$
$\mathit{SchedulableId}, \mathit{SchedulableIds}$

channel $\mathit{start\_toplevel\_sequencer} : \mathit{SchedulableID}$
channel $\mathit{set\_continue} : \mathit{SchedulableID} \times \mathbb{B}$

channelset $\mathit{CCSync} == \{\!| \mathit{get\_continue}, \mathit{set\_continue} |\!\}$
channelset $\mathit{TopLevelMissionSequencerFWChan} ==$
$\{\!| \mathit{start\_toplevel\_sequencer}, \mathit{end\_sequencer\_app}, \mathit{end\_methods},$
$\mathit{get\_continue}, \mathit{set\_continue} |\!\}$

C.5.17 MissionChan

section $\mathit{MissionChan}$ **parents** $\mathit{MissionFWChan}, \mathit{MissionMethChan}, \mathit{SchedulableMethChan}$

channelset $\mathit{MissionAppSync} == \{\!| \mathit{initializeCall}, \mathit{initializeRet}, \mathit{register},$
$\mathit{cleanupMissionCall}, \mathit{cleanupMissionRet}, \mathit{end\_mission\_app} |\!\}$

C.5.18 MissionFWChan

section $\mathit{MissionFWChan}$ **parents** $\mathit{scj\_prelude}, \mathit{MissionId}, \mathit{SchedulableId}$

channel $\mathit{start\_mission} : \mathit{MissionID} \times \mathit{SchedulableID}$
channel $\mathit{done\_mission} : \mathit{MissionID} \times \mathbb{B}$
channel $\mathit{done\_schedulables} : \mathit{MissionID}$
channel $\mathit{stop\_schedulables} : \mathit{MissionID}$
channel $\mathit{get\_activeSchedulables} : \mathit{MissionID} \times (\mathbb{F}\, \mathit{SchedulableID})$
channel $\mathit{schedulables\_stopped} : \mathit{MissionID}$
channel $\mathit{schedulables\_terminated} : \mathit{MissionID}$
channel $\mathit{end\_mission\_terminations} : \mathit{MissionID}$
channel $\mathit{end\_mission\_fw} : \mathit{MissionID}$
channel $\mathit{end\_mission\_app} : \mathit{MissionID}$
channel $\mathit{get\_missionTerminating} : \mathit{MissionID} \times \mathbb{B}$
channel $\mathit{set\_missionTerminating} : \mathit{MissionID} \times \mathbb{B}$

channelset $TerminateSync == \lbrace\!| schedulables\_terminated,$
$schedulables\_stopped, get\_activeSchedulables |\!\rbrace$
channelset $MTCSync == \lbrace\!| get\_missionTerminating,$
$set\_missionTerminating, end\_mission\_terminations |\!\rbrace$

C.5.19 MissionMethChan

section $MissionMethChan$ **parents** $scj\_prelude, MissionId, SchedulableId$

channel $initializeCall : MissionID$
channel $initializeRet : MissionID$
channel $cleanupMissionCall : MissionID$
channel $cleanupMissionRet : MissionID \times \mathbb{B}$
channel $requestTerminationCall : MissionID \times SchedulableID$
channel $requestTerminationRet : MissionID \times SchedulableID \times \mathbb{B}$
channel $terminationPendingCall : MissionID$
channel $terminationPendingRet : MissionID \times \mathbb{B}$
channel $missionactive : MissionID \times \mathbb{B}$

channelset $MissionMethChan ==$
$\lbrace\!| initializeCall, initializeRet, cleanupMissionCall, cleanupMissionRet,$
$requestTerminationCall, requestTerminationRet,$
$terminationPendingCall, terminationPendingRet |\!\rbrace$

C.5.20 SchedulableChan

section $SchedulableChan$ **parents** $MissionId, SchedulableId,$
$SchedulableFWChan, SchedulableMethChan$

C.5.21 SchedulableMethChan

section $SchedulableMethChan$ **parents** $MissionId, SchedulableId$

channel $register : SchedulableID \times MissionID$
channel $signalTerminationCall : SchedulableID$
channel $signalTerminationRet : SchedulableID$
channel $cleanupSchedulableCall : SchedulableID$
channel $cleanupSchedulableRet : SchedulableID$

C.5.22 SchedulableFWChan

section $SchedulableFWChan$ **parents** $MissionId, SchedulableId$

channel $activate\_schedulables : MissionID$
channel $done\_schedulable : SchedulableID$

C.5.23 SchedulableMissionSequencerChan

section $SchedulableMissionSequencerChan$ **parents**
$SchedulableMissionSequencerFWChan, MissionSequencerChan$

C.5.24 SchedulableMissionSequencerFWChan

section $SchedulableMissionSequencerFWChan$ **parents** $scj\_prelude,$
$MissionSequencerFWChan, SchedulableId, SchedulableIds$

channel $set\_continueBelow : SchedulableID \times \mathbb{B}$
channel $set\_continueAbove : SchedulableID \times \mathbb{B}$

channelset $CCSync == \{\!| get\_continue, set\_continueBelow, set\_continueAbove |\!\}$

channelset $SchedulableMissionSequencerFWChan ==$
$\{\!| end\_sequencer\_app, end\_methods, end\_terminations, get\_continue |\!\}$

C.5.25 HandlerChan

section $HandlerChan$ **parents** $HandlerFWChan, HandlerMethChan,$
$MissionFWChan, JTime$

C.5.26 HandlerFWChan

section $HandlerFWChan$ **parents** $scj\_prelude, SchedulableId, GlobalTypes, JTime$

channel $deschedule\_handler : SchedulableID$
channel $end\_releases : SchedulableID$
channel $release\_complete : SchedulableID$

C.5.27 HandlerMethChan

section $\mathit{HandlerMethChan}$ **parents** $\mathit{scj\_prelude}, \mathit{SchedulableId}, \mathit{GlobalTypes}$

channel $\mathit{handleAsyncEventCall} : \mathit{SchedulableID}$
channel $\mathit{handleAsyncEventRet} : \mathit{SchedulableID}$
channel $\mathit{release} : \mathit{SchedulableID}$

C.5.28 AperiodicEventHandlerChan

section $\mathit{AperiodicEventHandlerChan}$ **parents** $\mathit{HandlerChan}$,
$\mathit{HandlerMethChan}, \mathit{AperiodicLongEventHandlerMethChan}, \mathit{SchedulableId}$

channel $\mathit{end\_aperiodic\_app} : \mathit{SchedulableID}$
channelset $\mathit{APEHSync} == \{\!| \mathit{handleAsyncEventCall}, \mathit{handleAsyncEventRet},$
$\mathit{end\_aperiodicEventHandler\_app} |\!\}$
channelset $\mathit{DeadlineClockSync} == \{\!| \mathit{end\_releases}, \mathit{release\_complete} |\!\}$

C.5.29 AperiodicLongEventHandlerMethChan

section $\mathit{AperiodicLongEventHandlerMethChan}$ **parents** $\mathit{HandlerChan}$,
$\mathit{HandlerMethChan}, \mathit{MissionFWChan}, \mathit{JTime}$

channel $\mathit{releaseLong} : \mathit{SchedulableID} \times \mathbb{Z}$
channel $\mathit{handleAsyncLongEventCall} : \mathit{SchedulableID} \times \mathbb{Z}$
channel $\mathit{handleAsyncLongEventRet} : \mathit{SchedulableID}$

C.5.30 OneShotEventHandlerChan

section $\mathit{OneShotEventHandlerChan}$ **parents** $\mathit{HandlerChan}, \mathit{HandlerMethChan}$,
$\mathit{OneShotEventHandlerFWChan}, \mathit{OneShotEventHandlerMethChan}$

channelset $\mathit{STCSync} == \{\!| \mathit{get\_startTime}, \mathit{set\_startTime} |\!\}$
channelset $\mathit{MethodsSync} == \{\!| \mathit{end\_releases}, \mathit{reschedule\_handler}, \mathit{deschedule\_handler} |\!\}$
channelset $\mathit{ReleaseSync} == \{\!| \mathit{handleAsyncEventCall}, \mathit{reschedule\_handler}, \mathit{end\_releases},$
$\mathit{stop\_release}, \mathit{release} |\!\}$
channelset $\mathit{DeadlineSync} == \{\!| \mathit{handleAsyncEventCall}, \mathit{end\_releases},$
$\mathit{deschedule\_handler}, \mathit{release\_complete} |\!\}$
channelset $\mathit{OSEHAppSync} == \{\!| \mathit{descheduleCall}, \mathit{descheduleRet}, \mathit{scheduleNextRelease},$
$\mathit{getNextReleaseTimeCall}, \mathit{getNextReleaseTimeRet}, \mathit{end\_oneShot\_app} |\!\}$

C.5.31 OneShotEventHandlerFWChan

section *OneShotEventHandlerFWChan* **parents** *HandlerChan*, *HandlerMethChan*, *JTime*

channel $wait\_for\_start\_time : SchedulableID$
channel $end\_schedule : SchedulableID$
channel $reschedule\_handler : SchedulableID \times JTime$
channel $stop\_release : SchedulableID$
channel $end\_oneShot\_app : SchedulableID$
channel $get\_fireCount : SchedulableID \times \mathbb{Z}$
channel $increment\_fireCount : SchedulableID$
channel $decrement\_fireCount : SchedulableID$
channel $get\_startTime : SchedulableID \times JTime$
channel $set\_startTime : SchedulableID \times JTime$

C.5.32 OneShotEventHandlerMethChan

section *OneShotEventHandlerMethChan* **parents** *SchedulableId*, *JTime*

channel $getNextReleaseTimeCall : SchedulableID$
channel $getNextReleaseTimeRet : SchedulableID \times JTime$
channel $scheduleNextRelease : SchedulableID \times JTime$
channel $descheduleCall : SchedulableID$
channel $descheduleRet : SchedulableID \times \mathbb{B}$

C.5.33 PeriodicEventHandlerChan

section *PeriodicEventHandlerChan* **parents** *HandlerChan*, *HandlerMethChan*, *PeriodicEventHandlerFWChan*

channel $end\_periodic\_app : SchedulableID$

channelset $PEHSync ==$
$\{\!| handleAsyncEventCall, handleAsyncEventRet, end\_periodic\_app |\!\}$

C.5.34 PeriodicEventHandlerFWChan

section $PeriodicEventHandlerFWChan$ **parents** $HandlerChan, HandlerMethChan,$
$MissionFWChan, JTime$

channel $get\_missedReleases : SchedulableID \times \mathbb{Z}$
channel $increment\_missedReleases : SchedulableID$
channel $decrement\_missedReleases : SchedulableID$
channel $stop\_period : SchedulableID$
channel $periodic\_release\_complete : SchedulableID \times \mathbb{Z}$
channel $end\_periodic\_app : SchedulableID$
channel $get\_periodicTerminating : SchedulableID \times \mathbb{B}$
channel $set\_periodicTerminating : SchedulableID \times \mathbb{B}$

channelset $MRCSync == \{\!| get\_missedReleases, increment\_missedReleases,$
$decrement\_missedReleases |\!\}$
channelset $ReleaseSync == \{\!| release, stop\_period |\!\}$
channelset $PTCSYnc == \{\!| get\_periodicTerminating, set\_periodicTerminating |\!\}$

C.5.35 ManagedThreadChan

section $ManagedThreadChan$ **parents** $ManagedThreadFWChan,$
$ManagedThreadMethChan, SchedulableChan$

channelset $MTAppSync == \{\!| runCall, runRet, end\_managedThread\_app |\!\}$

C.5.36 ManagedThreadFWChan

section $ManagedThreadFWChan$ **parents** $SchedulableId$

channel $end\_managedThread\_app : SchedulableID$

C.5.37 ManagedThreadMethChan

section $ManagedThreadMethChan$ **parents** $SchedulableId$

channel $runCall : SchedulableID$
channel $runRet : SchedulableID$

C.6 Thread

section *ThreadFW* **parents** *scj_prelude*, *GlobalTypes*, *ThreadChan*, *ObjectFWChan*, *FrameworkChan*, *Priority*

process $ThreadFW \mathrel{\widehat=} thread : ThreadID;\ basePriority : PriorityLevel \bullet$ **begin**

state *State*

$priorityStack : \mathrm{seq}_1\ PriorityLevel$
$activePriority : PriorityLevel$
$interrupted : \mathbb{B}$

$activePriority = last\ priorityStack$

state *State*

Init

$\Delta State$

$priorityStack' = \langle basePriority \rangle$
$interrupted' = \mathbf{False}$

$$Execute \mathrel{\widehat=} \left(\left(\begin{array}{l}\left(\begin{array}{l}Priority \\ \lbrack\!\lbrack \{basePriority\} \mid \{interrupted\} \rbrack\!\rbrack \\ Interrupts\end{array}\right) \\ ||| \\ GetPriorityLevel\end{array}\right)\right) \triangle \left(\begin{array}{l}done\_toplevel\_sequencer \longrightarrow \\ \mathbf{Skip}\end{array}\right)$$

$$\begin{array}{l}Priority \mathrel{\widehat=} \\ \quad \mathbf{if}\ priorityStack = \langle basePriority \rangle \longrightarrow \\ \quad\quad IncreasePriority \quad [\!]\ priorityStack \neq \langle basePriority \rangle \longrightarrow \\ \quad\quad \left(\begin{array}{l}IncreasePriority \\ \Box DecreasePriority\end{array}\right) \\ \quad \mathbf{fi}\end{array}$$

$$\begin{array}{l}IncreasePriority \mathrel{\widehat=} \\ \quad raise\_thread\_priority\ .\ thread\ ?\ ceilingPriority \longrightarrow \\ \quad activePriority := ceilingPriority; \\ \quad IncreasePriority\end{array}$$

$$
\begin{array}{l}
\mathit{DecreasePriority} \mathrel{\widehat=} \\
\quad \mathit{lower\_thread\_priority} \,.\, \mathit{thread} \longrightarrow \\
\quad \mathit{activePriority} := \mathit{basePriority}; \\
\quad \mathit{DecreasePriority}
\end{array}
$$

$$
\begin{array}{l}
\mathit{Interrupts} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{Interrupt} \\
\quad [\![\varnothing \mid \varnothing]\!] \\
\mathit{IsInterrupted}
\end{array}\right) \\
\quad [\![\varnothing \mid \varnothing]\!] \\
\mathit{Interrupted}
\end{array}\right) \\
\quad [\![\varnothing \mid \{\!|\, \mathit{set\_interrupted}, \mathit{get\_interrupted} \,|\!\} \mid \varnothing]\!] \\
\mathit{InterruptedController}
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{Interrupt} \mathrel{\widehat=} \\
\quad \mathit{interrupt} \,.\, \mathit{thread} \longrightarrow \\
\quad \mathit{set\_interrupted} \,.\, \mathit{thread} \,!\, \mathit{True} \longrightarrow \mathbf{Skip}
\end{array}
$$

$$
\begin{array}{l}
\mathit{IsInterrupted} \mathrel{\widehat=} \\
\quad \mathit{isInterruptedCall} \,.\, \mathit{thread} \longrightarrow \\
\quad \mathit{get\_interrupted} \,.\, \mathit{thread} \,?\, \mathit{interrupted} \longrightarrow \\
\quad \mathit{isInterruptedRet} \,.\, \mathit{thread} \,!\, \mathit{interrupted} \longrightarrow \mathbf{Skip}
\end{array}
$$

$$
\begin{array}{l}
\mathit{Interrupted} \mathrel{\widehat=} \\
\quad \mathit{interruptedCall} \,.\, \mathit{thread} \longrightarrow \\
\quad \mathit{get\_interrupted} \,.\, \mathit{thread} \,?\, \mathit{interrupted} \longrightarrow \\
\quad \mathit{interruptedRet} \,.\, \mathit{thread} \,!\, \mathit{interrupted} \longrightarrow \\
\quad \mathit{set\_interrupted} \,.\, \mathit{thread} \,!\, \mathit{False} \longrightarrow \mathbf{Skip}
\end{array}
$$

$$
\begin{array}{l}
\mathit{InterruptedController} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mathit{get\_interrupted} \,.\, \mathit{thread} \,!\, \mathit{interrupted} \longrightarrow \\
\mathit{InterruptedController}
\end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
\mathit{set\_interrupted} \,.\, \mathit{thread} \,?\, \mathit{newInterrupted} \longrightarrow \\
\mathit{interrupted} := \mathit{newInterrupted}; \\
\mathit{InterruptedController}
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{GetPriorityLevel} \mathrel{\widehat=} \\
\quad \mathit{get\_priorityLevel} \,.\, \mathit{thread} \,?\, \mathit{object} \,!\, \mathit{activePriority} \longrightarrow \\
\quad \mathit{GetPriorityLevel}
\end{array}
$$

$$
\bullet\ (\mathit{Init} \,;\ \mathit{Execute}) \mathbin{\triangle} (\mathit{done\_toplevel\_sequencer} \longrightarrow \mathbf{Skip})
$$

end

C.7 Object

section *Object* **parents** *scj_prelude*, *GlobalTypes*, *ObjectChan*, *MissionChan*, *SchedulableChan*, *SchedulableId*, *MissionId*, *MissionIds*, *TopLevelMissionSequencerChan*, *HandlerChan*, *SafeletMethChan*, *FrameworkChan*, *PriorityQueue*, *Priority*, *ThreadChan*

process $ObjectFW \mathrel{\widehat=} object : ObjectID \bullet$ **begin**

state *State*

$waitQueue : PriorityQueue$
$lockedBy : ThreadID$
$locks : \mathbb{N}$
$previousLocks : ThreadMap$
$queueForLock : PriorityQueue$
$ceilingPriority : PriorityLevel$
$waitForObjectThreads : \mathbb{P}\ ThreadID$

$locks > 0 \Leftrightarrow lockedBy \neq nullThreadId$
$lockedBy \notin \operatorname{dom} previousLocks$
$lockedBy \notin ElementsOf(waitQueue)$
$lockedBy \notin ElementsOf(queueForLock)$
$waitForObjectThreads \subseteq ElementsOf(waitQueue)$

state *State*

Init

$State'$

$IsEmpty(queueForLock') = True$
$IsEmpty(waitQueue') = True$
$locks' = 0$
$previousLocks' = \varnothing$
$ceilingPriority' = MaxPriority$
$waitForObjectThreads' = \varnothing$

$$\begin{array}{l}
\textit{FullyUnlock} \\
\hline
\Delta State \\
lockedBy? : ThreadID \\
locks? : \mathbb{N}_1 \\
\hline
previousLocks' = previousLocks \oplus \{lockedBy? \mapsto locks?\} \\
lockedBy' = nullThreadId \\
locks' = 0 \\
waitQueue' = waitQueue \\
queueForLock' = queueForLock \\
ceilingPriority' = ceilingPriority \\
waitForObjectThreads' = waitForObjectThreads \\
\hline
\end{array}$$

$$\begin{array}{l}
\textit{AddToQueueForLock} \\
\hline
\Delta State \\
someThread? : ThreadID \\
priorityLevel? : PriorityLevel \\
\hline
someThread? \neq nullThreadId \\
someThread? \notin ElementsOf(queueForLock) \\
queueForLock' = AddToPriorityQueue(queueForLock, someThread?, priorityLevel?) \\
lockedBy' = lockedBy \\
locks' = locks \\
previousLocks' = previousLocks \\
waitQueue' = waitQueue \\
ceilingPriority' = ceilingPriority \\
waitForObjectThreads' = waitForObjectThreads \\
\hline
\end{array}$$

$$\begin{array}{l}
\textit{AssignEligible} \\
\hline
\Delta State \\
\hline
(queueForLock', lockedBy') = RemoveFromPriorityQueue(queueForLock) \\
lockedBy' \in \operatorname{dom} previousLocks \Rightarrow locks' = previousLocks(lockedBy') \\
lockedBy' \notin \operatorname{dom} previousLocks \Rightarrow locks' = 1 \\
previousLocks' = \{lockedBy\} \lhd previousLocks \\
waitQueue' = waitQueue \\
ceilingPriority' = ceilingPriority \\
waitForObjectThreads' = waitForObjectThreads \\
\hline
\end{array}$$

$AddToWaitQueue$

$\Delta State$
$someThread? : ThreadID$
$priorityLevel? : PriorityLevel$
$waitType? : WaitType$

$someThread? \neq nullThreadId$
$someThread? \notin ElementsOf(waitQueue)$
$waitQueue' = AddToPriorityQueue(waitQueue, someThread?, priorityLevel?)$
$lockedBy' = lockedBy$
$locks' = locks$
$previousLocks' = previousLocks$
$queueForLock' = queueForLock$
$ceilingPriority' = ceilingPriority$
$waitType? = waitForObject \Rightarrow$
$\quad waitForObjectThreads' = waitForObjectThreads \cup \{someThread?\}$
$waitType? = wait \Rightarrow waitForObjectThreads' = waitForObjectThreads$

$RemoveThreadFromWaitQueue$

$\Delta State$
$waitingThread? : ThreadID$
$priorityLevel? : PriorityLevel$

$waitingThread? \in \operatorname{ran}(waitQueue(priorityLevel?))$
$waitQueue' = RemoveThreadFromPriorityQueue(waitQueue, waitingThread?, priorityLevel?)$
$lockedBy' = lockedBy$
$locks' = locks$
$previousLocks' = previousLocks$
$ceilingPriority' = ceilingPriority$
$waitForObjectThreads' = waitForObjectThreads \setminus \{waitingThread?\}$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{RemoveMostEligigbleFromWaitQueue}} \\
\hline
\Delta State \\
notified! : ThreadID \\
waitType! : WaitType \\
\hline
(waitQueue', notified!) = RemoveFromPriorityQueue(waitQueue) \\
lockedBy' = lockedBy \\
locks' = locks \\
previousLocks' = previousLocks \\
queueForLock' = queueForLock \\
ceilingPriority' = ceilingPriority \\
notified! \in waitForObjectThreads \Rightarrow waitType! = waitForObject \\
notified! \notin waitForObjectThreads \Rightarrow waitType! = wait \\
waitForObjectThreads' = waitForObjectThreads \setminus \{notified!\} \\
\hline
\end{array}$$

$$\begin{array}{l}
Execute \mathrel{\widehat=} \\
\quad \mathbf{var}\ interruptedThreads : \mathbb{P}\ ThreadID \bullet \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
Monitor \\
[\![\varnothing \mid \\
\quad MonitorSync \mid \\
\{waitQueue, waitForObjectThreads\}]\!] \\
Synchronisation
\end{array}\right) \\
[\![\{waitQueue, waitForObjectThreads\} \mid \\
\quad MLCSync \mid \\
\{queueForLock, previousLocks, locks, lockedBy\}]\!] \\
MonitorLockController(interruptedThreads)
\end{array}\right) \\
\quad [\![\{waitQueue, waitForObjectThreads, \\
\quad queueForLock, previousLocks, locks, lockedBy\} \mid \\
\qquad CPCSync \mid \\
\quad \{ceilingPriority\}]\!] \\
CeilingPriorityController
\end{array}\right)
\end{array}$$

$$\begin{array}{l}
Monitor \mathrel{\widehat=} \\
\quad MonitorUnlocked
\end{array}$$

$$
\begin{array}{l}
\mathit{MonitorUnlocked} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mathit{startSyncMeth} \,.\, \mathit{object} \,?\, \mathit{someThread} \longrightarrow \\
\mathit{lock\_request} \,.\, \mathit{object} \,!\, \mathit{someThread} \longrightarrow \\
\mathit{MonitorUnlocked}
\end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
\mathit{lockAcquired} \,.\, \mathit{object} \,?\, \mathit{lockingThread} \longrightarrow \\
\mathit{get\_ceilingPriority} \,.\, \mathit{object} \,?\, \mathit{ceilPri} \longrightarrow \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{get\_priorityLevel} \,.\, \mathit{lockingThread} \,.\, \mathit{object} \,?\, \mathit{pri} : (\mathit{pri} \leq \mathit{ceilPri}) \longrightarrow \\
\mathit{raise\_thread\_priority} \,.\, \mathit{lockingThread} \,!\, \mathit{ceilPri} \longrightarrow \\
\mathit{MonitorLocked}(\mathit{lockingThread})
\end{array}\right) \\
\Box \\
\left(\begin{array}{l}
\mathit{get\_priorityLevel} \,.\, \mathit{lockingThread} \,.\, \mathit{object} \,?\, \mathit{pri} : (\mathit{pri} > \mathit{ceilPri}) \longrightarrow \\
\mathit{throw.ceilingViolationException} \longrightarrow \\
\mathbf{Chaos}
\end{array}\right)
\end{array}\right)
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{MonitorLocked} \mathrel{\widehat=} \mathbf{val}\, \mathit{lockedBy} : \mathit{ThreadID} \bullet \\
\quad \left(\begin{array}{l}
\mathit{startSyncMeth} \,.\, \mathit{object} \,.\, \mathit{lockedBy} \longrightarrow \\
\mathit{increment\_locks} \,.\, \mathit{object} \longrightarrow \\
\mathit{MonitorLocked}(\mathit{lockedBy})
\end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
\mathit{startSyncMeth} \,.\, \mathit{object} \,?\, \mathit{someThread} : (\mathit{someThread} \neq \mathit{lockedBy}) \longrightarrow \\
\mathit{lock\_request} \,.\, \mathit{object} \,!\, \mathit{someThread} \longrightarrow \\
\mathit{MonitorLocked}(\mathit{lockedBy})
\end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
\mathit{endSyncMeth} \,.\, \mathit{object} \,.\, \mathit{lockedBy} \longrightarrow \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{decrement\_locks} \,.\, \mathit{object} \,.\, 0 \longrightarrow \\
\mathit{lower\_thread\_priority} \,.\, \mathit{lockedBy} \longrightarrow \\
\mathit{MonitorUnlocked}
\end{array}\right) \\
\Box \\
\left(\begin{array}{l}
\mathit{decrement\_locks} \,.\, \mathit{object} \,?\, l : (l \neq 0) \longrightarrow \\
\mathit{MonitorLocked}(\mathit{lockedBy})
\end{array}\right)
\end{array}\right)
\end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
\mathit{unlock\_Monitor} \,.\, \mathit{object} \,?\, \mathit{unlockingThread} \longrightarrow \\
\mathit{fully\_unlock} \,.\, \mathit{object} \longrightarrow \\
\mathit{lower\_thread\_priority} \,.\, \mathit{unlockingThread} \longrightarrow \\
\mathit{MonitorUnlocked}
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{Synchronisation} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{WaitActions} \\
[\![\varnothing \mid \mathit{WaitSync} \mid \varnothing]\!] \\
\mathit{NotifyActions}
\end{array}\right) \\
\quad [\![\varnothing \mid \\
\qquad \mathit{WQSync} \mid \\
\quad \{ \mathit{waitQueue}, \mathit{waitForObjectThreads} \}]\!] \\
\mathit{WaitQueueController}
\end{array}\right) \\
[\![\{ \mathit{waitQueue}, \mathit{waitForObjectThreads} \} \mid \mathit{InterruptSync} \mid \varnothing]\!] \\
\mathit{Interrupt}
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{WaitActions} \mathrel{\widehat=} \\
\quad (\mathit{Wait} \interleave \mathit{TimedWait}) \interleave \mathit{WaitForObject}
\end{array}
$$

$$
\begin{array}{l}
\mathit{NotifyActions} \mathrel{\widehat=} \\
\quad \mathit{Notify} \interleave \mathit{NotifyAll}
\end{array}
$$

$$
\begin{array}{l}
\mathit{Wait} \mathrel{\widehat=} \\
\quad \mathit{waitCall} \,.\, \mathit{object} \,?\, \mathit{someThread} \longrightarrow \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{isInterruptedCall} \,.\, \mathit{someThread} \longrightarrow \\
\mathit{isInterruptedRet} \,.\, \mathit{someThread} \,.\, \mathit{False} \longrightarrow \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{get\_lockedBy} \,.\, \mathit{object} \,.\, \mathit{someThread} \longrightarrow \\
\mathit{get\_priorityLevel} \,.\, \mathit{someThread} \,.\, \mathit{object} \,?\, \mathit{priorityLevel} \longrightarrow \\
\mathit{add\_to\_wait} \,.\, \mathit{object} \,!\, \mathit{someThread} \,!\, \mathit{priorityLevel} \,!\, \mathit{wait} \longrightarrow \\
\mathit{unlock\_Monitor} \,.\, \mathit{object} \,!\, \mathit{someThread} \longrightarrow \\
\mathit{Wait}
\end{array}\right) \\
\Box \\
\left(\begin{array}{l}
\mathit{get\_lockedBy} \,.\, \mathit{object} \,?\, \mathit{lockedBy} : (\mathit{lockedBy} \neq \mathit{someThread}) \longrightarrow \\
\mathit{throw} \,.\, \mathit{illegalMonitorStateException} \longrightarrow \\
\mathbf{Chaos}
\end{array}\right)
\end{array}\right)
\end{array}\right) \\
\Box \\
\left(\begin{array}{l}
\mathit{isInterruptedCall} \,.\, \mathit{someThread} \longrightarrow \\
\mathit{isInterruptedRet} \,.\, \mathit{someThread} \,.\, \mathit{True} \longrightarrow \\
\mathit{throw.interruptedException} \longrightarrow \\
\mathbf{Chaos}
\end{array}\right)
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{TimedWait} \mathrel{\widehat=} \\
\quad \mathit{TimedWaitHandler} \\
\qquad [\![\varnothing \mid \{\!|\, \mathit{start\_timer} \,|\!\} \mid \varnothing]\!] \\
\quad (\interleave\, t : \mathit{ThreadID} \bullet \mathit{TimedWaitTimer}(t))
\end{array}
$$

$$
\begin{array}{l}
\mathit{TimedWaitHandler} \mathrel{\widehat{=}} \\
\quad \mathit{timedWaitCall} \,.\, \mathit{object} \,?\, \mathit{someThread} \,?\, \mathit{waitTime} \longrightarrow \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{get\_lockedBy} \,.\, \mathit{object} \,.\, \mathit{someThread} \longrightarrow \\
\mathbf{if}((\mathit{timeMillis}(\mathit{waitTime}) < 0) \lor (\mathit{timeNanos}(\mathit{waitTime}) < 0 \\
\quad \land\ \mathit{timeNanos}(\mathit{waitTime}) > \mathit{maxNanos})) \longrightarrow \\
\left(\begin{array}{l}
\mathit{throw} \,.\, \mathit{illegalArgumentException} \longrightarrow \\
\mathbf{Chaos}
\end{array} \right) \\
[\!]((\mathit{timeMillis}(\mathit{waitTime}) > 0) \land (\mathit{timeNanos}(\mathit{waitTime}) > 0) \\
\quad \land\ (\mathit{timeNanos}(\mathit{waitTime}) \leq \mathit{maxNanos})) \longrightarrow \\
\left(\begin{array}{l}
\mathit{get\_priorityLevel} \,.\, \mathit{someThread} \,.\, \mathit{object} \,?\, \mathit{priorityLevel} \longrightarrow \\
\mathit{add\_to\_wait} \,.\, \mathit{object} \,!\, \mathit{someThread} \,!\, \mathit{priorityLevel} \,!\, \mathit{wait} \longrightarrow \\
\mathit{start\_timer} \,.\, \mathit{object} \,!\, \mathit{someThread} \,!\, \mathit{priorityLevel} \,!\, \mathit{waitTime} \longrightarrow \\
\mathit{unlock\_Monitor} \,.\, \mathit{object} \,!\, \mathit{someThread} \longrightarrow \\
\mathit{TimedWaitHandler}
\end{array} \right) \\
\mathbf{fi}
\end{array} \right) \\
\quad \Box \\
\left(\begin{array}{l}
\mathit{get\_lockedBy} \,.\, \mathit{object} \,?\, \mathit{lockedBy} : (\mathit{lockedBy} \neq \mathit{someThread}) \longrightarrow \\
\mathit{throw} \,.\, \mathit{illegalMonitorStateException} \longrightarrow \\
\mathbf{Chaos}
\end{array} \right)
\end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{TimedWaitTimer} \mathrel{\widehat{=}} \mathbf{val}\ \mathit{waitingThread} : \mathit{ThreadID} \bullet \\
\quad \left(\begin{array}{l}
\mathit{start\_timer} \,.\, \mathit{object} \,.\, \mathit{waitingThread} \,?\, \mathit{priorityLevel} \,?\, \mathit{waitTime} \longrightarrow \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathbf{wait}\ \mathit{valueOf}(\mathit{waitTime}); \\
\mathit{remove\_from\_wait} \,.\, \mathit{object} \,!\, \mathit{waitingThread} \,!\, \mathit{priorityLevel} \longrightarrow \\
\mathit{waitRet} \,.\, \mathit{object} \,!\, \mathit{waitingThread} \longrightarrow \\
\mathbf{Skip}
\end{array} \right) \\
\Box \\
\left(\begin{array}{l}
\mathit{cancel\_wait\_timer} \,.\, \mathit{object} \,.\, \mathit{waitingThread} \longrightarrow \\
\mathbf{Skip}
\end{array} \right)
\end{array} \right) \\
; \\
\mathit{relock\_this} \,.\, \mathit{object} \,!\, \mathit{waitingThread} \longrightarrow \\
\mathit{TimedWaitTimer}(\mathit{waitingThread})
\end{array} \right)
\end{array} \right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
\mathit{cancel\_wait\_timer} \,.\, \mathit{object} \,.\, \mathit{waitingThread} \longrightarrow \\
\mathit{TimedWaitTimer}(\mathit{waitingThread})
\end{array} \right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
(\mathit{waitRet} \,.\, \mathit{object} \,.\, \mathit{waitingThread} \longrightarrow \mathit{TimedWaitTimer}(\mathit{waitingThread})) \\
\Box \\
(\mathit{waitForObjectRet} \,.\, \mathit{object} \,.\, \mathit{waitingThread} \,?\, w \longrightarrow \mathit{TimedWaitTimer}(\mathit{waitingThread}))
\end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
WaitForObject \mathrel{\widehat=} \\
\quad WaitForObjectHandler \\
\qquad [\![\varnothing \mid \{\!| start\_waitForObject\_timer |\!\} \mid \varnothing]\!] \\
\quad (\interleave\, t : ThreadID \bullet WaitForObjectTimer(t))
\end{array}
$$

$$
\begin{array}{l}
WaitForObjectHandler \mathrel{\widehat=} \\
\quad waitForObjectCall\,.\,object\,?\,someThread\,?\,waitTime \longrightarrow \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
get\_lockedBy\,.\,object\,.\,someThread \longrightarrow \\
\mathbf{if}\,((timeMillis(waitTime) < 0) \lor (timeNanos(waitTime) < 0)) \longrightarrow \\
\left(\begin{array}{l}
throw\,.\,illegalArgumentException \longrightarrow \\
\mathbf{Chaos}
\end{array} \right) \\
[\!]\,((timeMillis(waitTime) \geq 0) \land (timeNanos(waitTime) \geq 0)) \longrightarrow \\
\left(\begin{array}{l}
get\_priorityLevel\,.\,someThread\,.\,object\,?\,priorityLevel \longrightarrow \\
add\_to\_wait\,.\,object\,?\,someThread\,?\,priorityLevel\,!\,waitForObject \longrightarrow \\
start\_waitForObject\_timer\,.\,object\,!\,someThread \\
\quad !\,priorityLevel\,!\,waitTime \longrightarrow \\
unlock\_Monitor\,.\,object\,!\,someThread \longrightarrow \\
WaitForObjectHandler
\end{array} \right) \\
\mathbf{fi}
\end{array} \right) \\
\quad \Box \\
\left(\begin{array}{l}
get\_lockedBy\,.\,object\,?\,lockedBy : (lockedBy \neq someThread) \longrightarrow \\
throw\,.\,illegalMonitorStateException \longrightarrow \\
\mathbf{Chaos}
\end{array} \right)
\end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
WaitForObjectTimer \mathrel{\widehat=} \mathbf{val}\ waitingThread : ThreadID \bullet \\
\quad \left(\begin{array}{l}
start\_waitForObject\_timer\,.\,object\,?\,waitingThread\,?\,priorityLevel\,?\,waitTime \longrightarrow \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
remove\_from\_wait\,.\,object\,!\,waitingThread\,!\,priorityLevel \longrightarrow \\
waitForObjectRet\,.\,object\,!\,waitingThread\,!\,\mathbf{False} \longrightarrow \\
\mathbf{Skip}
\end{array} \right) \\
\Box \\
\left(\begin{array}{l}
cancel\_wait\_timer\,.\,object\,.\,waitingThread \longrightarrow \\
\mathbf{Skip}
\end{array} \right)
\end{array} \right) ; \\
relock\_this\,.\,object\,!\,waitingThread \longrightarrow \\
WaitForObjectTimer(waitingThread)
\end{array} \right)
\end{array} \right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
cancel\_wait\_timer\,.\,object\,.\,waitingThread \longrightarrow \\
WaitForObjectTimer(waitingThread)
\end{array} \right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
(waitRet\,.\,object\,?\,n \longrightarrow WaitForObjectTimer(waitingThread)) \\
\Box \\
(waitForObjectRet\,.\,object\,?\,n\,?\,w \longrightarrow WaitForObjectTimer(waitingThread))
\end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{Notify} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mathit{notify} \,.\, \mathit{object} \,?\, \mathit{someThread} \longrightarrow \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{get\_lockedBy} \,.\, \mathit{object} \,.\, \mathit{someThread} \longrightarrow \\
\left(\begin{array}{l}
\mathbf{if}\ \mathit{IsEmpty}(\mathit{waitQueue}) = \mathit{False} \longrightarrow \\
\big(\mathit{ResumeThread}\ ;\ \mathit{Notify}\big) \\
[]\ \mathit{IsEmpty}(\mathit{waitQueue}) = \mathit{True} \longrightarrow \\
\quad \mathit{Notify} \\
\mathbf{fi}
\end{array}\right)
\end{array}\right) \\
\Box \\
\left(\begin{array}{l}
\mathit{get\_lockedBy} \,.\, \mathit{object} \,?\, \mathit{lockedBy} : (\mathit{lockedBy} \neq \mathit{someThread}) \longrightarrow \\
\mathit{throw} \,.\, \mathit{illegalMonitorStateException} \longrightarrow \mathbf{Chaos}
\end{array}\right)
\end{array}\right)
\end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
(\mathit{waitRet} \,.\, \mathit{object} \,?\, n \longrightarrow \mathit{Notify}) \\
\Box \\
(\mathit{waitForObjectRet} \,.\, \mathit{object} \,?\, n \,?\, w \longrightarrow \mathit{Notify})
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{ResumeThread} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mathit{removed\_thread} \,.\, \mathit{object} \,?\, \mathit{notified} \,.\, \mathit{wait} \longrightarrow \\
\mathit{cancel\_wait\_timer} \,.\, \mathit{object} \,!\, \mathit{notified} \longrightarrow \\
\mathit{relock\_this} \,.\, \mathit{object} \,!\, \mathit{notified} \longrightarrow \\
\mathit{waitRet} \,.\, \mathit{object} \,!\, \mathit{notified} \longrightarrow \mathbf{Skip}
\end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
\mathit{removed\_thread} \,.\, \mathit{object} \,?\, \mathit{notified} \,.\, \mathit{waitForObject} \longrightarrow \\
\mathit{cancel\_wait\_timer} \,.\, \mathit{object} \,!\, \mathit{notified} \longrightarrow \\
\mathit{relock\_this} \,.\, \mathit{object} \,!\, \mathit{notified} \longrightarrow \\
\mathit{waitForObjectRet} \,.\, \mathit{object} \,!\, \mathit{notified} \,!\, \mathbf{True} \longrightarrow \mathbf{Skip}
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{NotifyAll} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mathit{notifyAll} \,.\, \mathit{object} \,?\, \mathit{someThread} \longrightarrow \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{get\_lockedBy} \,.\, \mathit{object} \,.\, \mathit{someThread} \longrightarrow \\
\mathit{NotifyAllHandler}\ ;\ \mathit{NotifyAll}
\end{array}\right) \\
\Box \\
\left(\begin{array}{l}
\mathit{get\_lockedBy} \,.\, \mathit{object} \,?\, \mathit{lockedBy} : (\mathit{lockedBy} \neq \mathit{someThread}) \longrightarrow \\
\mathit{throw} \,.\, \mathit{illegalMonitorStateException} \longrightarrow \mathbf{Chaos}
\end{array}\right)
\end{array}\right)
\end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
(\mathit{waitRet} \,.\, \mathit{object} \,?\, n \longrightarrow \mathit{NotifyAll}) \\
\Box \\
(\mathit{waitForObjectRet} \,.\, \mathit{object} \,?\, n \,?\, w \longrightarrow \mathit{NotifyAll})
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{NotifyAllHandler} \mathrel{\widehat=} \mathbf{var}\ \mathit{notified} : \mathit{ThreadID} \bullet \\
\quad \mathbf{if}\ \mathit{IsEmpty}(\mathit{waitQueue}) = \mathit{False} \longrightarrow \left(\begin{array}{l}\mathit{ResumeThread};\\ \mathit{NotifyAllHandler}\end{array}\right) \\
\quad [\!]\ \mathit{IsEmpty}(\mathit{waitQueue}) = \mathit{True} \longrightarrow \mathbf{Skip} \\
\quad \mathbf{fi}
\end{array}
$$

$$
\begin{array}{l}
\mathit{WaitQueueController} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}\mathit{add\_to\_wait}\,.\,\mathit{object}\,?\,\mathit{someThread}\,?\,\mathit{priorityLevel}\,?\,\mathit{waitType} \longrightarrow \\ \mathit{AddToWaitQueue}; \\ \mathit{WaitQueueController}\end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l}\mathit{remove\_from\_wait}\,.\,\mathit{object}\,?\,\mathit{waitingThread}\,?\,\mathit{priorityLevel} \longrightarrow \\ \mathit{RemoveThreadFromWaitQueue}; \\ \mathit{WaitQueueController}\end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l}\mathit{IsEmpty}(\mathit{waitQueue}) = \mathit{False} \,\&\, \\ \mathbf{var}\ \mathit{notified} : \mathit{ThreadID};\ \mathit{waitType} : \mathit{WaitType} \bullet \\ \mathit{RemoveMostEligigbleFromWaitQueue}; \\ \mathit{removed\_thread}\,.\,\mathit{object}\,!\,\mathit{notified}\,!\,\mathit{waitType} \longrightarrow \\ \mathit{WaitQueueController}\end{array}\right) \\
\quad \Box \\
\quad \left(\mathit{get\_waitQueue}\,.\,\mathit{object}\,!\,\mathit{waitQueue} \longrightarrow \mathit{WaitQueueController}\right) \\
\quad \Box \\
\quad \left(\mathit{get\_waitForObjectThreads}\,.\,\mathit{object}\,!\,\mathit{waitForObjectThreads} \longrightarrow \mathit{WaitQueueController}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{Interrupt} \mathrel{\widehat=} \\
\quad \mathit{interrupt}\,?\,\mathit{waitingThread} \longrightarrow \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{get\_waitQueue}\,.\,\mathit{object}\,?\,\mathit{gotWait} : (\mathit{waitingThread} \in \mathit{ElementsOf}(\mathit{gotWait})) \longrightarrow \\
\mathit{cancel\_wait\_timer}\,.\,\mathit{object}\,!\,\mathit{waitingThread} \longrightarrow \\
\mathit{get\_priorityLevel}\,.\,\mathit{waitingThread}\,.\,\mathit{object}\,?\,\mathit{priorityLevel} \longrightarrow \\
\mathit{remove\_from\_wait}\,.\,\mathit{object}\,!\,\mathit{waitingThread}\,!\,\mathit{priorityLevel} \longrightarrow \\
\mathit{relock\_this}\,.\,\mathit{object}\,!\,\mathit{waitingThread} \longrightarrow \\
\left(\begin{array}{l}
\left(\begin{array}{l}\mathit{get\_waitForObjectThreads}\,.\,\mathit{object}\,?\,\mathit{wfot} : (\mathit{waitingThread} \notin \mathit{wfot}) \longrightarrow \\ \mathit{waitRet}\,.\,\mathit{object}\,!\,\mathit{waitingThread} \longrightarrow \mathbf{Skip}\end{array}\right) \\
\Box \\
\left(\begin{array}{l}\mathit{get\_waitForObjectThreads}\,.\,\mathit{object}\,?\,\mathit{wfot} : (\mathit{waitingThread} \in \mathit{wfot}) \longrightarrow \\ \mathit{waitForObjectRet}\,.\,\mathit{object}\,!\,\mathit{waitingThread}\,!\,\mathbf{True} \longrightarrow \mathbf{Skip}\end{array}\right)
\end{array}\right); \\
\mathit{Interrupt}
\end{array}\right) \\
\Box \\
\left(\begin{array}{l}\mathit{get\_waitQueue}\,.\,\mathit{object}\,?\,\mathit{gotWait} : (\mathit{waitingThread} \notin \mathit{ElementsOf}(\mathit{gotWait})) \longrightarrow \\ \mathit{Interrupt}\end{array}\right)
\end{array}\right)
\end{array}
$$

$$MonitorLockController \mathrel{\widehat{=}} \mathbf{val}\, interruptedThreads : \mathbb{P}\, ThreadID \bullet$$

$$\left(\begin{array}{l} lock\_request \,.\, object \,?\, someThread \longrightarrow \\ get\_priorityLevel \,.\, someThread \,.\, object \,?\, priorityLevel \longrightarrow \\ AddToQueueForLock; \\ MonitorLockController(interruptedThreads) \end{array}\right)$$

$$\Box$$

$$\left(\begin{array}{l} relock\_this \,.\, object \,?\, someThread \longrightarrow \\ get\_priorityLevel \,.\, someThread \,.\, object \,?\, priorityLevel \longrightarrow \\ \left(\begin{array}{l} AddToQueueForLock; \\ \left(\begin{array}{l} \left(\begin{array}{l} isInterruptedCall \,.\, someThread \longrightarrow \\ isInterruptedRet \,.\, someThread \,.\, False \longrightarrow \\ MonitorLockController(interruptedThreads) \end{array}\right) \\ \Box \\ \left(\begin{array}{l} isInterruptedCall \,.\, someThread \longrightarrow \\ isInterruptedRet \,.\, someThread \,.\, True \longrightarrow \\ interruptedThreads := interruptedThreads \cup \{someThread\}; \\ MonitorLockController(interruptedThreads) \end{array}\right) \end{array}\right) \end{array}\right) \end{array}\right)$$

$$\Box$$

$$\left(\begin{array}{l} IsEmpty(queueForLock) = False \land lockedBy = nullThreadId \,\&\, \\ \left(\begin{array}{l} AssignEligible; \\ lockAcquired \,.\, object \,.\, lockedBy \longrightarrow \\ \mathbf{if}\ lockedBy \in interruptedThreads \longrightarrow \left(\begin{array}{l} throw.interruptedException \longrightarrow \\ \mathbf{Chaos} \end{array}\right) \\ [\!]\, lockedBy \notin interruptedThreads \longrightarrow \left(MonitorLockController(interruptedThreads)\right) \\ \mathbf{fi} \end{array}\right) \end{array}\right)$$

$$\Box$$

$$\left(\begin{array}{l} get\_lockedBy \,.\, object \,!\, lockedBy \longrightarrow \\ MonitorLockController(interruptedThreads) \end{array}\right)$$

$$\Box$$

$$\left(\begin{array}{l} increment\_locks \,.\, object \longrightarrow \\ locks := locks + 1; \\ MonitorLockController(interruptedThreads) \end{array}\right)$$

$$\Box$$

$$\left(\begin{array}{l} decrement\_locks \,.\, object \,!\, (locks - 1) \longrightarrow \\ \left(\begin{array}{l} locks := locks - 1; \\ \left(\begin{array}{l} \mathbf{if}\ locks = 0 \longrightarrow \left(\begin{array}{l} lockedBy := nullThreadId; \\ MonitorLockController(interruptedThreads) \end{array}\right) \\ [\!]\, locks \neq 0 \longrightarrow \quad MonitorLockController(interruptedThreads) \\ \mathbf{fi} \end{array}\right) \end{array}\right) \end{array}\right)$$

$$\Box$$

$$\left(\begin{array}{l} fully\_unlock \,.\, object \longrightarrow \\ FullyUnlock; \\ MonitorLockController(interruptedThreads) \end{array}\right)$$

$$
\begin{array}{l}
\mathit{CeilingPriorityController} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mathit{setCeilingPriority}\,?\,\mathit{mission}\,!\,\mathit{object}\,?\,\mathit{priority} \longrightarrow \\
\mathit{ceilingPriority} := \mathit{priority}; \\
\mu X \bullet (\mathit{get\_ceilingPriority}\,.\,\mathit{object}\,!\,\mathit{ceilingPriority} \longrightarrow X)
\end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
\mathit{get\_ceilingPriority}\,.\,\mathit{object}\,!\,\mathit{ceilingPriority} \longrightarrow \\
\mathit{CeilingPriorityController}
\end{array}\right)
\end{array}
$$

$$\bullet\ (\mathit{Init}\ ;\ \ \mathit{Execute}) \bigtriangleup (\mathit{done\_toplevel\_sequencer} \longrightarrow \mathbf{Skip})$$

end

C.8 SafeletFW

section $SafeletFW$ **parents** $scj\_prelude, SchedulableId, SchedulableIds, SafeletChan,$
$TopLevelMissionSequencerChan, FrameworkChan, SchedulableChan$

process $SafeletFW \mathrel{\widehat=}$ **begin**

state S_State

$globallyRegistered : \mathbb{F}\ SchedulableID$
$topLevelSequencer : SchedulableID$

S_Init

S_State'

$globallyRegistered' = \varnothing$
$topLevelSequencer' = nullSequencerId$

state S_State

```
$InitializeApplication \mathrel{\widehat=}$
  $initializeApplicationCall \longrightarrow$
  $initializeApplicationRet \longrightarrow$
  **Skip**
```

```
$Execute \mathrel{\widehat=}$
  $GetSequencerMeth;$
  **if** $topLevelSequencer \neq nullSequencerId \longrightarrow$
  $\left(\begin{array}{l} start\_toplevel\_sequencer\ .\ topLevelSequencer \longrightarrow \\ Methods \end{array}\right)$
  $[\!]\ topLevelSequencer = nullSequencerId \longrightarrow$
    **Skip**
  **fi**
```

```
$GetSequencerMeth \mathrel{\widehat=}$
  $getSequencerCall \longrightarrow$
  $getSequencerRet\ ?\ sequencer \longrightarrow$
  $topLevelSequencer := sequencer$
```

$$Methods \mathrel{\widehat=} \left(\begin{array}{l}\left(\begin{array}{l}Register;\\ Methods\end{array}\right)\\ \Box\\ \left(\begin{array}{l}Deregister;\\ Methods\end{array}\right)\\ \Box\\ \left(\begin{array}{l}done\_toplevel\_sequencer \longrightarrow\\ \mathbf{Skip}\end{array}\right)\end{array}\right)$$

$$Register \mathrel{\widehat=}$$
$$\left(\begin{array}{l}register\,?\,schedulable : (schedulable \notin globallyRegistered)\,?\,mission \longrightarrow\\ \left(\begin{array}{l}globallyRegistered := globallyRegistered \cup \{schedulable\};\\ checkSchedulable\,.\,mission\,!\,\mathbf{True} \longrightarrow\\ \mathbf{Skip}\end{array}\right)\end{array}\right)$$
$$\Box$$
$$\left(\begin{array}{l}register\,?\,schedulable : (schedulable \in globallyRegistered)\,?\,mission \longrightarrow\\ checkSchedulable\,.\,mission\,!\,\mathbf{False} \longrightarrow\\ \mathbf{Skip}\end{array}\right)$$

$$\begin{array}{l}Deregister \mathrel{\widehat=}\\ \quad deregister\,?\,schedulables \longrightarrow\\ \quad globallyRegistered := (globallyRegistered \setminus schedulables);\\ \quad \mathbf{Skip}\end{array}$$

$$\bullet \left(S\_Init\ ;\ InitializeApplication\ ;\ Execute\right)$$

end

C.9 TopLevelMissionSequencerFW

section $\mathit{TopLevelMissionSequencerFW}$ **parents** $\mathit{TopLevelMissionSequencerChan}$,
$\mathit{MissionId}, \mathit{MissionMethChan}, \mathit{SchedulableId}, \mathit{MissionFWChan}, \mathit{FrameworkChan}$

process $\mathit{TopLevelMissionSequencerFW} \mathrel{\widehat{=}} \mathit{sequencer} : \mathit{SchedulableID} \bullet$ **begin**

state $\mathit{TLMS\_State}$

$\mathit{currentMission} : \mathit{MissionID}$

$\mathit{continue} : \mathbb{B}$

state $\mathit{TLMS\_State}$

$\mathit{TLMS\_Init}$

$\mathit{TLMS\_State}'$

$\mathit{continue}' = \mathbf{True}$

$\mathit{currentMission}' = \mathit{nullMissionId}$

$$\begin{array}{l}
\mathit{Start} \mathrel{\widehat{=}} \\
\quad \mathit{start\_toplevel\_sequencer} \,.\, \mathit{sequencer} \longrightarrow \\
\quad \mathbf{Skip}
\end{array}$$

$$\begin{array}{l}
\mathit{Execute} \mathrel{\widehat{=}} \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{RunMission}; \\
\mathit{end\_methods} \,.\, \mathit{sequencer} \longrightarrow \\
\mathbf{Skip}
\end{array}\right) \\
[\![\{ \mathit{currentMission} \} \mid \{\!|\, \mathit{end\_methods} \,|\!\} \mid \varnothing]\!] \\
\mathit{Methods}
\end{array}\right) \\
[\![\varnothing \mid \mathit{CCSync} \mid \{ \mathit{continue} \}]\!] \\
\mathit{ContinueController}
\end{array}\right)
\end{array}$$

$$\begin{array}{l}
\mathit{RunMission} \mathrel{\widehat{=}} \\
\quad \mathit{GetNextMission}; \\
\quad \mathit{StartMission}; \\
\quad \mathit{Continue}
\end{array}$$

$$\begin{array}{l}
\mathit{GetNextMission} \mathrel{\widehat{=}} \\
\quad \mathit{getNextMissionCall} \,.\, \mathit{sequencer} \longrightarrow \\
\quad \mathit{getNextMissionRet} \,.\, \mathit{sequencer} \,?\, \mathit{next} \longrightarrow \\
\quad \mathit{currentMission} := \mathit{next}
\end{array}$$

$$
\begin{array}{l}
StartMission \mathrel{\widehat=} \\
\quad \mathbf{if}\ currentMission \neq nullMissionId \longrightarrow \\
\qquad \left(\begin{array}{l}
start\_mission\,.\,currentMission\,.\,sequencer \longrightarrow \\
done\_mission\,.\,currentMission\,?\,returnedcontinue \longrightarrow \\
set\_continue\,.\,sequencer\,!\,returnedcontinue \longrightarrow \\
\mathbf{Skip}
\end{array}\right) \\
\quad [\!]\ currentMission = nullMissionId \longrightarrow \\
\qquad \left(\begin{array}{l}
set\_continue\,.\,sequencer\,!\,\mathbf{False} \longrightarrow \\
\mathbf{Skip}
\end{array}\right) \\
\quad \mathbf{fi}
\end{array}
$$

$$
\begin{array}{l}
Continue \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
get\_continue\,.\,sequencer\,?\,continue : (continue = \mathbf{True}) \longrightarrow \\
RunMission
\end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
get\_continue\,.\,sequencer\,?\,continue : (continue = \mathbf{False}) \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
Methods \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
SequenceTerminationPending; \\
Methods
\end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
end\_methods\,.\,sequencer \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
SequenceTerminationPending \mathrel{\widehat=} \\
\quad sequenceTerminationPendingCall\,.\,sequencer \longrightarrow \\
\quad get\_continue\,.\,sequencer\,?\,continue \longrightarrow \\
\quad sequenceTerminationPendingRet\,.\,sequencer\,!\,continue \longrightarrow \\
\quad \mathbf{Skip}
\end{array}
$$

$$
\begin{array}{l}
ContinueController \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
get\_continue\,.\,sequencer\,!\,continue \longrightarrow \\
ContinueController
\end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
set\_continue\,.\,sequencer\,?\,newContinue \longrightarrow \\
continue := newContinue; \\
ContinueController
\end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
end\_methods\,.\,sequencer \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}
$$

$Finish \mathrel{\widehat=}$

$$\left(\begin{array}{l} done\_toplevel\_sequencer \longrightarrow \\ end\_sequencer\_app\,.\,sequencer \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$\bullet\ TLMS\_Init\ ;\ Start\ ;\ Execute\ ;\ Finish$

end

C.10 MissionFW

section $MissionFW$ **parents** $scj\_prelude, SafeletMethChan, MissionId,$
$SchedulableId, MissionChan, SchedulableChan, FrameworkChan, ServicesChan$

process $MissionFW \mathrel{\widehat=} mission : MissionID \bullet$ **begin**

state M_State

$registeredSchedulables : \mathbb{F}\ SchedulableID$
$activeSchedulables : \mathbb{F}\ SchedulableID$
$missionTerminating : \mathbb{B}$
$applicationTerminating : \mathbb{B}$
$controllingSequencer : SchedulableID$

state M_State

M_Init

M_State'

$registeredSchedulables' = \varnothing$
$activeSchedulables' = \varnothing$
$missionTerminating =$ **False**
$applicationTerminating =$ **False**
$controllingSequencer = nullSequencerId$

$AddSchedulable$

ΔM_State
$s? : SchedulableID$

$s? \notin registeredSchedulables$
$registeredSchedulables' = registeredSchedulables \cup \{s?\}$
$activeSchedulables' = activeSchedulables$
$missionTerminating' = missionTerminating$
$applicationTerminating' = applicationTerminating$
$controllingSequencer' = controllingSequencer$

$$Start \mathrel{\widehat=} \left(\begin{array}{l} start\_mission\ .\ mission\ ?\ mySequencer \longrightarrow \\ controllingSequencer := mySequencer \end{array}\right) \\ \Box \\ \left(\begin{array}{l} done\_toplevel\_sequencer \longrightarrow \\ applicationTerminating := \mathbf{True} \end{array}\right)$$

$$
\begin{array}{l}
\mathit{InitializePhase} \mathrel{\widehat{=}} \\
\quad \mathit{initializeCall} \,.\, \mathit{mission} \longrightarrow \\
\quad \mathit{Initialize}
\end{array}
$$

$$
\begin{array}{l}
\mathit{Initialize} \mathrel{\widehat{=}} \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l} \mathit{Register}; \\ \mathit{Initialize} \end{array}\right) \\
\Box \\
\left(\begin{array}{l} \mathit{SetCeilingPriority}; \\ \mathit{Initialize} \end{array}\right) \\
\Box \\
\left(\begin{array}{l} \mathit{initializeRet} \,.\, \mathit{mission} \longrightarrow \\ \mathbf{Skip} \end{array}\right)
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{Register} \mathrel{\widehat{=}} \\
\quad \mathit{register} \,?\, s \,!\, \mathit{mission} \longrightarrow \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l} \mathit{checkSchedulable} \,.\, \mathit{mission} \,?\, \mathit{check} : (\mathit{check} = \mathbf{True}) \longrightarrow \\ \mathit{AddSchedulable} \end{array}\right) \\
\Box \\
\left(\begin{array}{l} \mathit{checkSchedulable} \,.\, \mathit{mission} \,?\, \mathit{check} : (\mathit{check} = \mathbf{False}) \longrightarrow \\ \mathit{throw}.\mathit{illegalStateException} \longrightarrow \\ \mathbf{Chaos} \end{array}\right)
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{RegisterException} \mathrel{\widehat{=}} \\
\quad \mathit{register} \,?\, s \,!\, \mathit{mission} \longrightarrow \\
\quad \mathit{throw}.\mathit{illegalStateException} \longrightarrow \\
\quad \mathbf{Chaos}
\end{array}
$$

$$
\begin{array}{l}
\mathit{SetCeilingPriority} \mathrel{\widehat{=}} \\
\quad \mathit{setCeilingPriority} \,.\, \mathit{mission} \,?\, o \,?\, p \longrightarrow \\
\quad \mathbf{Skip}
\end{array}
$$

$$
\begin{array}{l}
\mathit{SetCeilingPriorityException} \mathrel{\widehat{=}} \\
\quad \mathit{setCeilingPriority} \,.\, \mathit{mission} \,?\, o \,?\, p \longrightarrow \\
\quad \mathit{throw}.\mathit{illegalStateException} \longrightarrow \\
\quad \mathbf{Chaos}
\end{array}
$$

$$
\begin{array}{l}
\mathit{MissionPhase} \mathrel{\widehat{=}} \\
\quad \mathit{Execute} \\
\qquad [\![\{ \mathit{registeredSchedulables}, \mathit{activeSchedulables}, \mathit{missionTerminating}, \\
\quad \mathit{applicationTerminating}, \mathit{controllingSequencer} \} \mid \{\!| \mathit{done\_schedulables} |\!\} \mid \varnothing]\!] \\
\quad \mathit{Exceptions}
\end{array}
$$

$$
\begin{array}{l}
\mathit{Execute} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\textbf{if } \mathit{registeredSchedulables} = \varnothing \longrightarrow \\
\quad \left(\begin{array}{l} \mathit{done\_schedulables} \,.\, \mathit{mission} \longrightarrow \\ \textbf{Skip} \end{array} \right) \\
[\!] \ \mathit{registeredSchedulables} \neq \varnothing \longrightarrow \\
\quad \left(\begin{array}{l}
\mathit{activate\_schedulables} \,.\, \mathit{mission} \longrightarrow \\
\mathit{activeSchedulables} := \mathit{registeredSchedulables}; \\
\left(\begin{array}{l}
\mathit{TerminateAndDone} \\
\quad [\![\{ \mathit{activeSchedulables} \} \mid \\
\qquad \{\!| \mathit{stop\_schedulables}, \mathit{done\_schedulables} |\!\} \mid \\
\quad \{ \mathit{missionTerminating} \}]\!] \\
\mathit{Methods}
\end{array} \right)
\end{array} \right) \\
\textbf{fi}
\end{array} \right) \setminus \{\!| \mathit{done\_schedulables} |\!\}
\end{array}
$$

$$
\begin{array}{l}
\mathit{TerminateAndDone} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{SignalTermination} \\
\quad [\![\varnothing \mid \mathit{TerminateSync} \mid \{ \mathit{activeSchedulables} \}]\!] \\
\mathit{DoneSchedulables}
\end{array} \right); \\
\mathit{done\_schedulables} \,.\, \mathit{mission} \longrightarrow \\
\textbf{Skip}
\end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{SignalTermination} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mathit{stop\_schedulables} \,.\, \mathit{mission} \longrightarrow \\
\mathit{get\_activeSchedulables} \,.\, \mathit{mission} \,?\, \mathit{schedulablesToStop} \longrightarrow \\
\mathit{StopSchedulables}(\mathit{schedulablesToStop}); \\
\mathit{schedulables\_stopped} \,.\, \mathit{mission} \longrightarrow \\
\textbf{Skip}
\end{array} \right) \\
\quad \triangle (\mathit{schedulables\_stopped} \,.\, \mathit{mission} \longrightarrow \textbf{Skip})
\end{array}
$$

$$
\begin{array}{l}
\mathit{StopSchedulables} \mathrel{\widehat=} \textbf{val } \mathit{schedulablesToStop} : \mathbb{F}\, \mathit{SchedulableID} \bullet \\
\quad \left(\begin{array}{l}
|||\ s : \mathit{schedulablesToStop} \bullet \\
\qquad \mathit{signalTerminationCall} \,.\, s \longrightarrow \\
\qquad \mathit{signalTerminationRet} \,.\, s \longrightarrow \\
\qquad \textbf{Skip}
\end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{DoneSchedulables} \mathrel{\widehat{=}} \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\Box\, \mathit{schedulable} : \mathit{activeSchedulables} \bullet \\
\mathit{done\_schedulable} \,.\, \mathit{schedulable} \longrightarrow \\
\mathit{activeSchedulables} := \mathit{activeSchedulables} \setminus \{\mathit{schedulable}\}; \\
\mathbf{Skip}
\end{array}\right); \\
\mathbf{if}\ \mathit{activeSchedulables} = \emptyset \longrightarrow \\
\qquad \left(\begin{array}{l}
\mathit{schedulables\_stopped} \,.\, \mathit{mission} \longrightarrow \\
\mathbf{Skip}
\end{array}\right) \\
[\!]\ \mathit{activeSchedulables} \neq \emptyset \longrightarrow \\
\qquad \mathit{DoneSchedulables} \\
\mathbf{fi}
\end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
\mathit{get\_activeSchedulables} \,.\, \mathit{mission} \,!\, \mathit{activeSchedulables} \longrightarrow \\
\mathit{DoneSchedulables}
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{Methods} \mathrel{\widehat{=}} \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{RequestTerminationMeth} \\
\llbracket \emptyset \mid \{\!|\, \mathit{end\_mission\_terminations} \,|\!\} \mid \emptyset \rrbracket \\
\mathit{TerminationPendingMeth}
\end{array}\right) \\
\llbracket \emptyset \mid \mathit{MTCSync} \mid \{\mathit{missionTerminating}\} \rrbracket \\
\mathit{MissionTerminatingController}
\end{array}\right) \\
\quad \llbracket \{\mathit{missionTerminating}\} \mid \{\!|\, \mathit{end\_mission\_terminations} \,|\!\} \mid \emptyset \rrbracket \\
\quad \left(\begin{array}{l}
\mathit{done\_schedulables} \,.\, \mathit{mission} \longrightarrow \\
\mathit{end\_mission\_terminations} \,.\, \mathit{mission} \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{RequestTerminationMeth} \mathrel{\widehat{=}} \\
\quad \left(\begin{array}{l}
\mathit{end\_mission\_terminations} \,.\, \mathit{mission} \longrightarrow \\
\mathbf{Skip}
\end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
\Box\, \mathit{schedulable} : \mathit{registeredSchedulables} \bullet \\
\quad \mathit{requestTerminationCall} \,.\, \mathit{mission} \,.\, \mathit{schedulable} \longrightarrow \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{get\_missionTerminating} \,.\, \mathit{mission} ? \mathit{mT} : (\mathit{mT} = \mathbf{False}) \longrightarrow \\
\mathit{set\_missionTerminating} \,.\, \mathit{mission} \,!\, \mathbf{True} \longrightarrow \\
\mathit{stop\_schedulables} \,.\, \mathit{mission} \longrightarrow \\
\mathit{requestTerminationRet} \,.\, \mathit{mission} \,.\, \mathit{schedulable} \,.\, \mathbf{False} \longrightarrow \\
\mathit{RequestTerminationMeth}
\end{array}\right) \\
\Box \\
\left(\begin{array}{l}
\mathit{get\_missionTerminating} \,.\, \mathit{mission} ? \mathit{mT} : (\mathit{mT} = \mathbf{True}) \longrightarrow \\
\mathit{requestTerminationRet} \,.\, \mathit{mission} \,.\, \mathit{schedulable} \,.\, \mathbf{True} \longrightarrow \\
\mathit{RequestTerminationMeth}
\end{array}\right)
\end{array}\right)
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
TerminationPendingMeth \mathrel{\widehat=} \\
\quad \left(\begin{array}{l} end\_mission\_terminations \,.\, mission \longrightarrow \\ \mathbf{Skip} \end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l} terminationPendingCall \,.\, mission \longrightarrow \\ get\_missionTerminating \,.\, mission \,?\, missionTerminating \longrightarrow \\ terminationPendingRet \,.\, mission \,!\, missionTerminating \longrightarrow \\ TerminationPendingMeth \end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
MissionTerminatingController \mathrel{\widehat=} \\
\quad \left(\begin{array}{l} get\_missionTerminating \,.\, mission \,!\, missionTerminating \longrightarrow \\ MissionTerminatingController \end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l} set\_missionTerminating \,.\, mission \,?\, newMissionTerminating \longrightarrow \\ missionTerminating := newMissionTerminating; \\ MissionTerminatingController \end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l} end\_mission\_terminations \,.\, mission \longrightarrow \\ \mathbf{Skip} \end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
CleanupPhase \mathrel{\widehat=} \\
\quad Cleanup \\
\qquad [\![\{ registeredSchedulables, activeSchedulables, \\
\qquad\qquad missionTerminating, applicationTerminating, \\
\qquad\quad controllingSequencer \} \mid \{\!| done\_schedulables |\!\} \mid \varnothing]\!] \\
\quad Exceptions
\end{array}
$$

$$
\begin{array}{l}
Cleanup \mathrel{\widehat=} \\
\quad \left(\begin{array}{l} deregister!registeredSchedulables \longrightarrow \\ CleanupSchedulables; \\ cleanupMissionCall \,.\, mission \longrightarrow \\ cleanupMissionRet \,.\, mission \,?\, continueSequencer \longrightarrow \\ Finish(continueSequencer) \end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
CleanupSchedulables \mathrel{\widehat=} \\
\quad \left(\begin{array}{l} ||| \; s : registeredSchedulables \bullet \\ \quad cleanupSchedulableCall \,.\, s \longrightarrow \\ \quad cleanupSchedulableRet \,.\, s \longrightarrow \\ \quad \mathbf{Skip} \end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
Finish \mathrel{\widehat=} \mathbf{val}\; continueSequencer : \mathbb{B} \bullet \\
\quad end\_mission\_app \,.\, mission \longrightarrow \\
\quad done\_mission \,.\, mission \,!\, continueSequencer \longrightarrow \\
\quad \mathbf{Skip}
\end{array}
$$

$$Exceptions \mathrel{\widehat=}$$
$$\left(\begin{array}{l} RegisterException \\ ||| \\ SetCeilingPriorityException \end{array}\right)$$
$$\Box$$
$$\left(\begin{array}{l} done\_schedulables \,.\, mission \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$\bullet \left(\begin{array}{l} \mu X \bullet M\_Init \;;\; Start; \\ \left(\begin{array}{l} \mathbf{if}\ applicationTerminating = \mathbf{False} \longrightarrow \\ \qquad \left(InitializePhase \;;\; MissionPhase \;;\; CleanupPhase \;;\; X\right) \\ [\!]\ applicationTerminating = \mathbf{True} \longrightarrow \\ \qquad \left(\begin{array}{l} end\_mission\_app \,.\, mission \longrightarrow \\ \mathbf{Skip} \end{array}\right) \\ \mathbf{fi} \end{array}\right) \end{array}\right)$$

end

C.11 SchedulableMissionSequencerFW

section *SchedulableMissionSequencerFW* **parents** *SchedulableMissionSequencerChan*, *SchedulableChan*, *MissionIds*, *MissionChan*, *SchedulableId*, *scj_prelude*, *SafeletMethChan*, *FrameworkChan*

process *SchedulableMissionSequencerFW* $\widehat{=}$ *sequencer* : *SchedulableID* • **begin**

state *SMS_State*

$currentMission : MissionID$
$continueAbove : \mathbb{B}$
$continueBelow : \mathbb{B}$
$controllingMission : MissionID$
$applicationTerminating : \mathbb{B}$

state *SMS_State*

SMS_Init

SMS_State'

$continueAbove' = \textbf{True}$
$continueBelow' = \textbf{True}$
$applicationTerminating' = \textbf{False}$
$currentMission' = nullMissionId$
$controllingMission' = nullMissionId$

GetContinue

ΞSMS_State
$continue! : \mathbb{B}$

$continueAbove = \textbf{True} \land continueBelow = \textbf{True} \Rightarrow continue! = \textbf{True}$

$$Start \ \widehat{=}$$
$$\begin{pmatrix} Register; \\ Activate \end{pmatrix}$$
$$\Box$$
$$\begin{pmatrix} done\_toplevel\_sequencer \longrightarrow \\ applicationTerminating := \textbf{True} \end{pmatrix}$$
$$\Box$$
$$\begin{pmatrix} activate\_schedulables \, ? \, someMissionID \longrightarrow \\ Start \end{pmatrix}$$

$$\begin{array}{l} Register \mathrel{\widehat=} \\ \quad register\,.\,sequencer\,?\,mID \longrightarrow \\ \quad controllingMission := mID \end{array}$$

$$\begin{array}{l} Activate \mathrel{\widehat=} \\ \quad activate\_schedulables.controllingMission \longrightarrow \\ \quad \mathbf{Skip} \end{array}$$

$$\begin{array}{l} Execute \mathrel{\widehat=} \\ \quad \left(\begin{array}{l} \left(\begin{array}{l} \left(\begin{array}{l} RunMission; \\ end\_methods\,.\,sequencer \longrightarrow \\ \mathbf{Skip} \end{array}\right) \\ [\![\{ currentMission \} \mid \{\!| \, end\_methods \, |\!\} \mid \varnothing]\!] \\ Methods \end{array}\right) \\ [\![\varnothing \mid CCSync \mid \{ continueAbove, continueBelow \}]\!] \\ ContinueController \end{array}\right); \\ \quad done\_schedulable\,.\,sequencer \longrightarrow \mathbf{Skip} \end{array}$$

$$\begin{array}{l} RunMission \mathrel{\widehat=} \\ \quad GetNextMission; \\ \quad StartMission; \\ \quad Continue \end{array}$$

$$\begin{array}{l} GetNextMission \mathrel{\widehat=} \\ \quad getNextMissionCall\,.\,sequencer \longrightarrow \\ \quad getNextMissionRet\,.\,sequencer\,?\,next \longrightarrow \\ \quad currentMission := next \end{array}$$

$$\begin{array}{l} StartMission \mathrel{\widehat=} \\ \quad \mathbf{if} \;\; currentMission \neq nullMissionId \longrightarrow \\ \qquad \left(\begin{array}{l} start\_mission\,.\,currentMission\,.\,sequencer \longrightarrow \\ initializeRet\,.\,currentMission \longrightarrow \\ \left(\begin{array}{l} SignalTermination \\ \quad [\![\varnothing \mid \{\!| \, end\_terminations \, |\!\} \mid \varnothing]\!] \\ \left(\begin{array}{l} done\_mission\,.\,currentMission\,?\,continueReturn \longrightarrow \\ set\_continueBelow\,.\,sequencer\,!\,continueReturn \longrightarrow \\ end\_terminations\,.\,sequencer \longrightarrow \\ \mathbf{Skip} \end{array}\right) \end{array}\right) \end{array}\right) \\ \quad [\!] \; currentMission = nullMissionId \longrightarrow \\ \qquad \left(\begin{array}{l} set\_continueBelow\,.\,sequencer\,!\,\mathbf{False} \longrightarrow \\ \mathbf{Skip} \end{array}\right) \\ \quad \mathbf{fi} \end{array}$$

$$
\begin{array}{l}
\mathit{Continue} \mathrel{\widehat=} \\
\quad \left(\mathit{get\_continue} \,.\, \mathit{sequencer} \,?\, \mathit{continue} : (\mathit{continue} = \mathbf{True}) \longrightarrow \mathit{RunMission} \right) \\
\quad \Box \\
\quad \left(\mathit{get\_continue} \,.\, \mathit{sequencer} \,?\, \mathit{continue} : (\mathit{continue} = \mathbf{False}) \longrightarrow \mathbf{Skip} \right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{SignalTermination} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\left(\mathit{end\_terminations} \,.\, \mathit{sequencer} \longrightarrow \mathbf{Skip} \right) \\
\Box \\
\left(\begin{array}{l}
\mathit{signalTerminationCall} \,.\, \mathit{sequencer} \longrightarrow \\
\mathit{set\_continueAbove} \,.\, \mathit{sequencer} \,!\, \mathbf{False} \longrightarrow \\
\mathit{requestTerminationCall} \,.\, \mathit{currentMission} \,.\, \mathit{sequencer} \longrightarrow \\
\mathit{requestTerminationRet} \,.\, \mathit{currentMission} \,.\, \mathit{sequencer} \,?\, \mathit{isTerminating} \longrightarrow \\
\mathit{signalTerminationRet} \,.\, \mathit{sequencer} \longrightarrow \mathbf{Skip}
\end{array} \right) ; \\
\mathit{end\_terminations} \,.\, \mathit{sequencer} \longrightarrow \mathbf{Skip}
\end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{Methods} \mathrel{\widehat=} \\
\quad \left(\mathit{SequenceTerminationPending} \,;\, \mathit{Methods} \right) \\
\quad \Box \\
\quad \left(\mathit{end\_methods} \,.\, \mathit{sequencer} \longrightarrow \mathbf{Skip} \right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{SequenceTerminationPending} \mathrel{\widehat=} \\
\quad \mathit{sequenceTerminationPendingCall} \,.\, \mathit{sequencer} \longrightarrow \\
\quad \mathit{get\_continue} \,.\, \mathit{sequencer} \,?\, \mathit{continue} \longrightarrow \\
\quad \mathit{sequenceTerminationPendingRet} \,.\, \mathit{sequencer} \,!\, \mathit{continue} \longrightarrow \\
\quad \mathbf{Skip}
\end{array}
$$

$$
\begin{array}{l}
\mathit{ContinueController} \mathrel{\widehat=} \mathbf{var}\ \mathit{continue} : \mathbb{B} \bullet \\
\quad \left(\begin{array}{l}
\mathit{GetContinue} \,;\, \mathit{get\_continue} \,.\, \mathit{sequencer} \,!\, \mathit{continue} \longrightarrow \\
\mathit{ContinueController}
\end{array} \right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
\mathit{set\_continueBelow} \,.\, \mathit{sequencer} \,?\, \mathit{newContinueBelow} \longrightarrow \\
\mathit{continueBelow} := \mathit{newContinueBelow}; \\
\mathit{ContinueController}
\end{array} \right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
\mathit{set\_continueAbove} \,.\, \mathit{sequencer} \,?\, \mathit{newContinueAbove} \longrightarrow \\
\mathit{continueAbove} := \mathit{newContinueAbove}; \\
\mathit{ContinueController}
\end{array} \right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
\mathit{end\_methods} \,.\, \mathit{sequencer} \longrightarrow \\
\mathbf{Skip}
\end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{Cleanup} \mathrel{\widehat=} \\
\quad \mathit{cleanupSchedulableCall} \,.\, \mathit{sequencer} \longrightarrow \\
\quad \mathit{cleanupSchedulableRet} \,.\, \mathit{sequencer} \longrightarrow \\
\quad \mathit{Finish}
\end{array}
$$

$Finish \mathrel{\widehat=}$

$\quad done\_schedulable\,.\,sequencer \longrightarrow$

$\quad \mathbf{Skip}$

$$\bullet \left(\begin{array}{l} \mu X \bullet SMS\_Init\ ;\ Start; \\ \left(\begin{array}{l} \mathbf{if}\ applicationTerminating = \mathbf{False} \longrightarrow \\ \qquad \left(Execute\ ;\ Cleanup\ ;\ X\right) \\ \talloblong\ applicationTerminating = \mathbf{True} \longrightarrow \\ \qquad \left(\begin{array}{l} end\_sequencer\_app\,.\,sequencer \longrightarrow \\ \mathbf{Skip} \end{array}\right) \\ \mathbf{fi} \end{array}\right) \end{array}\right)$$

end

C.12 PeriodicEventHandlerFW

section $\mathit{PeriodicEventHandlerFW}$ **parents** $\mathit{MissionChan}, \mathit{SchedulableChan}, \mathit{SchedulableId}, \mathit{MissionId}, \mathit{MissionIds}, \mathit{TopLevelMissionSequencerChan}, \mathit{PeriodicEventHandlerChan}, \mathit{SafeletMethChan}, \mathit{FrameworkChan}, \mathit{PeriodicParameters}$

process $\mathit{PeriodicEventHandlerFW} \mathrel{\widehat=}$
$\mathit{schedulable} : \mathit{SchedulableID};\ \mathit{periodicParameters} : \mathit{PeriodicParameters} \bullet$ **begin**

state $\mathit{PEH\_State}$

$\mathit{controllingMission} : \mathit{MissionID}$
$\mathit{applicationTerminating} : \mathbb{B}$
$\mathit{period} : \mathit{JTime}$
$\mathit{startTime} : \mathit{JTime}$
$\mathit{deadline} : \mathit{JTime}$
$\mathit{deadlineMissHandler} : \mathit{SchedulableID}$
$\mathit{missedReleases} : \mathbb{N}$
$\mathit{periodicTerminating} : \mathbb{B}$

$\mathit{valueOf}(\mathit{deadline}) \leq \mathit{valueOf}(\mathit{period})$

state $\mathit{PEH\_State}$

$\mathit{PEH\_Init}$

$\mathit{PEH\_State}'$

$\mathit{controllingMission}' = \mathit{nullMissionId}$
$\mathit{applicationTerminating}' = \mathbf{False}$
$\mathit{periodicTerminating}' = \mathbf{False}$
$\mathit{period}' = \mathit{periodOf}(\mathit{periodicParameters})$
$\mathit{startTimeOf}(\mathit{periodicParameters}) = \mathit{NULL}$
$\quad \Rightarrow \mathit{startTime}' = \mathit{time}\,(0, 0)$
$\mathit{startTimeOf}(\mathit{periodicParameters}) \neq \mathit{NULL}$
$\quad \Rightarrow \mathit{startTime}' = \mathit{startTimeOf}(\mathit{periodicParameters})$
$\mathit{deadlineOfPeriodic}(\mathit{periodicParameters}) = \mathit{NULL}$
$\quad \Rightarrow \mathit{deadline}' = \mathit{period}'$
$\mathit{deadlineOfPeriodic}(\mathit{periodicParameters}) \neq \mathit{NULL}$
$\quad \Rightarrow \mathit{deadline}' = \mathit{deadlineOfPeriodic}(\mathit{periodicParameters})$
$\mathit{missedReleases}' = 0$
$\mathit{deadlineMissHandler}' = \mathit{missHandlerOfPeriodic}(\mathit{periodicParameters})$

$$
\begin{array}{l}
\mathit{Start} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l} \mathit{Register}; \\ \mathit{Activate} \end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l} \mathit{activate\_schedulables}?\mathit{someMissionID} \longrightarrow \\ \mathit{Start} \end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l} \mathit{done\_toplevel\_sequencer} \longrightarrow \\ \mathit{applicationTerminating} := \mathbf{True} \end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{Register} \mathrel{\widehat=} \\
\quad \mathit{register}\,.\,\mathit{schedulable}\,?\,\mathit{missionID} \longrightarrow \\
\quad \mathit{controllingMission} := \mathit{missionID}
\end{array}
$$

$$
\begin{array}{l}
\mathit{Activate} \mathrel{\widehat=} \\
\quad \mathit{activate\_schedulables}\,.\,\mathit{controllingMission} \longrightarrow \\
\quad \mathbf{Skip}
\end{array}
$$

$$
\begin{array}{l}
\mathit{Execute} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathbf{wait}\ \mathit{valueOf}(\mathit{startTime}); \\
\mathbf{if}\ \mathit{deadlineMissHandler} \neq \mathit{nullSchedulableId} \longrightarrow \\
\quad \mathit{RunningWithDeadlineDetection} \\
[\!]\ \mathit{deadlineMissHandler} = \mathit{nullSchedulableId} \longrightarrow \\
\quad \mathit{Running} \\
\mathbf{fi}
\end{array}\right) \\
\Box \\
\left(\begin{array}{l} \mathit{end\_releases}\,.\,\mathit{schedulable} \longrightarrow \\ \mathbf{Skip} \end{array}\right)
\end{array}\right) \\
\quad [\![\{\mathit{startTime}\} \mid \{\!|\ \mathit{stop\_period}\ |\!\} \mid \varnothing]\!] \\
\mathit{SignalTermination}
\end{array}\right) \\
\quad [\![\{\mathit{startTime}\} \mid \mathit{PTCSYnc} \mid \varnothing]\!] \\
\mathit{PeriodicTerminatingController}
\end{array}
$$

$$
\begin{array}{l}
\mathit{Running} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mathit{PeriodicClock} \\
\quad [\![\varnothing \mid \mathit{ReleaseSync} \mid \{\mathit{missedReleases}\}]\!] \\
\mathit{Release}(0)
\end{array}\right)
\end{array}
$$

$$
\mathit{RunningWithDeadlineDetection} \widehat{=} \\
\left(\begin{array}{l}
\mathit{Running} \\
\quad [\![\{ \mathit{missedReleases} \} \mid \mathit{ReleaseSync} \mid \varnothing]\!] \\
\mathit{DeadlineClock}(0)
\end{array}\right)
$$

$$
\mathit{PeriodicClock} \widehat{=} \\
\mathit{release} \,.\, \mathit{schedulable} \longrightarrow \\
\mu X \bullet \left(\begin{array}{l}
\left(\begin{array}{l}
\mathbf{wait}\ \mathit{valueOf}(\mathit{period}); \\
\mathit{release} \,.\, \mathit{schedulable} \longrightarrow \\
X
\end{array}\right) \\
\Box \\
\left(\begin{array}{l}
\mathit{end\_releases} \,.\, \mathit{schedulable} \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}\right)
$$

$$
\mathit{Release} \widehat{=} \mathbf{val}\ \mathit{index} : \mathbb{N} \bullet \\
\mathbf{if}\ \mathit{missedReleases} = 0 \longrightarrow \\
\quad \left(\begin{array}{l}
\mathit{release} \,.\, \mathit{schedulable} \longrightarrow \\
\mathit{handleAsyncEventCall} \,.\, \mathit{schedulable} \longrightarrow \\
\mathbf{Skip}
\end{array}\right) \\
[\!]\ \mathit{missedReleases} \neq 0 \longrightarrow \\
\quad \left(\begin{array}{l}
\mathit{handleAsyncEventCall} \,.\, \mathit{schedulable} \longrightarrow \\
\mathit{missedReleases} := \mathit{missedReleases} - 1; \\
\mathbf{Skip}
\end{array}\right) \\
\mathbf{fi}\,; \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{handleAsyncEventRet} \,.\, \mathit{schedulable} \longrightarrow \\
\mathit{periodic\_release\_complete} \,.\, \mathit{schedulable} \,.\, \mathit{index} \longrightarrow \\
\mathbf{Skip}
\end{array}\right) \\
\quad [\![\varnothing \mid \{\!| \mathit{handleAsyncEventRet} |\!\} \mid \varnothing]\!] \\
\left(\mu X \bullet \left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{release} \,.\, \mathit{schedulable} \longrightarrow \\
\mathit{missedReleases} := \mathit{missedReleases} + 1; \\
X
\end{array}\right) \\
\Box \\
\left(\begin{array}{l}
\mathit{handleAsyncEventRet} \,.\, \mathit{schedulable} \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}\right)\right)
\end{array}\right); \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{get\_periodicTerminating} \,.\, \mathit{schedulable}\,?\,\mathit{pehTerm} : (\mathit{pehTerm} = \mathbf{False}) \longrightarrow \\
\mathit{Release}(\mathit{index} + 1)
\end{array}\right) \\
\Box \\
\left(\begin{array}{l}
\mathit{get\_periodicTerminating} \,.\, \mathit{schedulable}\,?\,\mathit{pehTerm} : (\mathit{pehTerm} = \mathbf{True}) \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}\right)
$$

$$DeadlineClock \mathrel{\widehat=} \mathbf{val}\, index : \mathbb{N} \bullet$$

$$\left(\begin{array}{l}\left(\begin{array}{l}\left(\begin{array}{l}\mathbf{wait}\ valueOf(deadline);\\ release \,.\, deadlineMissHandler \longrightarrow\\ periodic\_release\_complete \,.\, schedulable \,.\, index \longrightarrow\\ \mathbf{Skip}\end{array}\right)\\ \Box\\ \left(\begin{array}{l}periodic\_release\_complete \,.\, schedulable \,.\, index \longrightarrow\\ \mathbf{Skip}\end{array}\right)\end{array}\right)\\ ||| \left(\left(\begin{array}{l}\mathbf{wait}\ valueOf(period);\\ DeadlineClock(index + 1)\end{array}\right)\right)\end{array}\right)$$

$$\triangle \left(\begin{array}{l}end\_releases \,.\, schedulable \longrightarrow\\ periodic\_release\_complete \,.\, schedulable \,?\, index \longrightarrow\\ \mathbf{Skip}\end{array}\right)$$

$$\begin{array}{l}SignalTermination \mathrel{\widehat=}\\ \quad signalTerminationCall \,.\, schedulable \longrightarrow\\ \quad set\_periodicTerminating \,.\, schedulable \,!\, \mathbf{True} \longrightarrow\\ \quad end\_releases \,.\, schedulable \longrightarrow\\ \quad signalTerminationRet \,.\, schedulable \longrightarrow\\ \quad done\_schedulable \,.\, schedulable \longrightarrow\\ \quad \mathbf{Skip}\end{array}$$

$$\begin{array}{l}Cleanup \mathrel{\widehat=}\\ \quad cleanupSchedulableCall \,.\, schedulable \longrightarrow\\ \quad cleanupSchedulableRet \,.\, schedulable \longrightarrow\\ \quad \mathbf{Skip}\end{array}$$

$$\begin{array}{l}PeriodicTerminatingController \mathrel{\widehat=}\\ \quad \left(\begin{array}{l}get\_periodicTerminating \,.\, schedulable \,!\, periodicTerminating \longrightarrow\\ PeriodicTerminatingController\end{array}\right)\\ \quad \Box\\ \quad \left(\begin{array}{l}set\_periodicTerminating \,.\, schedulable \,?\, newPeriodicTerminating \longrightarrow\\ periodicTerminating := newPeriodicTerminating;\\ PeriodicTerminatingController\end{array}\right)\end{array}$$

$$\bullet \left(\mu X \bullet \left(\begin{array}{l}PEH\_Init \,;\ Start;\\ \left(\begin{array}{l}\mathbf{if}\ applicationTerminating = \mathbf{False} \longrightarrow\\ \qquad \left(Execute \,;\ Cleanup \,;\ X\right)\\ [\!]\ applicationTerminating = \mathbf{True} \longrightarrow\\ \qquad \left(end\_periodic\_app \,.\, schedulable \longrightarrow \mathbf{Skip}\right)\\ \mathbf{fi}\end{array}\right)\end{array}\right)\right)$$

end

C.13 AperiodicEventHandlerFW

section $\mathit{AperiodicEventHandlerFW}$ **parents** $\mathit{MissionChan}, \mathit{SchedulableChan}, \mathit{SchedulableId}, \mathit{MissionId}, \mathit{MissionIds}, \mathit{TopLevelMissionSequencerChan}, \mathit{SafeletMethChan}, \mathit{FrameworkChan}, \mathit{AperiodicEventHandlerChan}, \mathit{AperiodicParameters}$

process $\mathit{AperiodicEventHandlerFW} \mathrel{\widehat=}$
$\mathit{schedulable} : \mathit{SchedulableID};\ \mathit{aperiodicType} : \mathit{AperiodicType};$
$\mathit{aperiodicParameters} : \mathit{AperiodicParameters}$ ● **begin**

state $\mathit{APEH\_State}$

$\mathit{controllingMission} : \mathit{MissionID}$
$\mathit{applicationTerminating} : \mathbb{B}$
$\mathit{pending} : \mathbb{B}$
$\mathit{data} : \mathbb{Z}$
$\mathit{deadline} : \mathit{JTime}$
$\mathit{deadlineMissHandler} : \mathit{SchedulableID}$

state $\mathit{APEH\_State}$

$\mathit{APEH\_Init}$

$\mathit{APEH\_State}'$

$\mathit{controllingMission}' = \mathit{nullMissionId}$
$\mathit{applicationTerminating}' = \mathbf{False}$
$\mathit{pending}' = \mathbf{False}$
$\mathit{deadline}' = \mathit{deadlineOfAperiodic}(\mathit{aperiodicParameters})$
$\mathit{deadlineMissHandler}' = \mathit{missHandlerOfAperiodic}(\mathit{aperiodicParameters})$

$$
\begin{array}{l}
\mathit{Start} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l} \mathit{Register}; \\ \mathit{Activate} \end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l} \mathit{activate\_schedulables}?\mathit{someMissionID} \longrightarrow \\ \mathit{Start} \end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l} \mathit{done\_toplevel\_sequencer} \longrightarrow \\ \mathit{applicationTerminating} := \mathbf{True} \end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{Register} \mathrel{\widehat=} \\
\quad \mathit{register} \,.\, \mathit{schedulable} \,?\, \mathit{missionID} \longrightarrow \\
\quad \mathit{controllingMission} := \mathit{missionID}
\end{array}
$$

$$\begin{array}{l}
Activate \mathrel{\widehat=} \\
\quad activate\_schedulables\,.\,controllingMission \longrightarrow \\
\quad \mathbf{Skip}
\end{array}$$

$$\begin{array}{l}
Execute \mathrel{\widehat=} \\
\quad \mathbf{if}\ deadlineMissHandler! = nullSchedulableId \longrightarrow \\
\qquad \left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathbf{if}\ aperiodicType = aperiodic \longrightarrow \\
Ready \\
\mathrel{[\!]}\ aperiodicType = aperiodicLong \longrightarrow \\
ReadyLong \\
\mathbf{fi}
\end{array}\right) \\
\llbracket \{pending, data\} \mid \{\!|\ end\_releases\ |\!\} \mid \varnothing \rrbracket \\
SignalTermination
\end{array}\right) \\
\llbracket \{pending, data\} \mid \\
\quad DeadlineClockSync \cup \{\!|\ release.schedulable, releaseLong.schedulable\ |\!\} \\
\mid \varnothing \rrbracket \\
\left(\begin{array}{l}
\left(\left(\begin{array}{l}
release\,.\,schedulable \longrightarrow \mathbf{Skip} \\
\square \\
releaseLong\,.\,schedulable?data \longrightarrow \mathbf{Skip}
\end{array}\right) ;\ DeadlineClock\right) \\
\triangle \left(end\_releases.schedulable \longrightarrow \mathbf{Skip}\right)
\end{array}\right)
\end{array}\right) \\
\quad \mathrel{[\!]}\ deadlineMissHandler = nullSchedulableId \longrightarrow \\
\qquad \left(\begin{array}{l}
\left(\begin{array}{l}
\mathbf{if}\ aperiodicType = aperiodic \longrightarrow \\
Ready \\
\mathrel{[\!]}\ aperiodicType = aperiodicLong \longrightarrow \\
ReadyLong \\
\mathbf{fi}
\end{array}\right) \\
\llbracket \{pending, data\} \mid \{\!|\ end\_releases\ |\!\} \mid \varnothing \rrbracket \\
SignalTermination
\end{array}\right) \\
\quad \mathbf{fi}
\end{array}$$

$$\begin{array}{l}
DeadlineClock \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\mathbf{wait}\ valueOf(deadline); \\
release\,.\,deadlineMissHandler \longrightarrow DeadlineClock
\end{array}\right) \\
\square \\
\left(release\_complete\,.\,schedulable \longrightarrow DeadlineClock\right)
\end{array}\right) \\
\quad \triangle \left(end\_releases\,.\,schedulable \longrightarrow \mathbf{Skip}\right)
\end{array}$$

$$\mathit{Ready} \mathrel{\widehat=}$$
$$\left(\begin{array}{l}\mathit{release.schedulable}\longrightarrow\\ \mathit{handleAsyncEventCall}\,.\,\mathit{schedulable}\longrightarrow \mathit{Release}\end{array}\right)$$
$$\Box$$
$$\left(\mathit{end\_releases}\,.\,\mathit{schedulable}\longrightarrow \mathbf{Skip}\right)$$

$$\mathit{ReadyLong} \mathrel{\widehat=}$$
$$\left(\begin{array}{l}\mathit{releaseLong}\,.\,\mathit{schedulable}\,?\,\mathit{longData}\longrightarrow\\ \mathit{data} := \mathit{longData};\\ \mathit{handleAsyncLongEventCall}\,.\,\mathit{schedulable}\,.\,\mathit{data}\longrightarrow \mathit{ReleaseLong}\end{array}\right)$$
$$\Box$$
$$\left(\mathit{end\_releases}\,.\,\mathit{schedulable}\longrightarrow \mathbf{Skip}\right)$$

$$\mathit{SignalTermination} \mathrel{\widehat=}$$
$$\left(\begin{array}{l}\mathit{signalTerminationCall}\,.\,\mathit{schedulable}\longrightarrow\\ \mathit{end\_releases}\,.\,\mathit{schedulable}\longrightarrow\\ \mathit{signalTerminationRet}\,.\,\mathit{schedulable}\longrightarrow\\ \mathit{done\_schedulable}\,.\,\mathit{schedulable}\longrightarrow \mathbf{Skip}\end{array}\right)$$

$$\mathit{Release} \mathrel{\widehat=}$$
$$\left(\begin{array}{l}\mathit{release.schedulable}\longrightarrow\\ \mathit{pending} := \mathbf{True};\\ \mathit{Release}\end{array}\right)$$
$$\Box$$
$$\left(\begin{array}{l}\mathit{handleAsyncEventRet.schedulable}\longrightarrow\\ \mathbf{if}\ \mathit{pending} = \mathbf{True}\longrightarrow\\ \qquad\left(\begin{array}{l}\mathit{pending} := \mathbf{False};\\ \mathit{release\_complete}\,.\,\mathit{schedulable}\longrightarrow\\ \mathit{handleAsyncEventCall.schedulable}\longrightarrow\\ \mathit{Release}\end{array}\right)\\ []\,\mathit{pending} = \mathbf{False}\longrightarrow\\ \qquad \mathit{Ready}\\ \mathbf{fi}\end{array}\right)$$
$$\Box$$
$$\left(\mathit{end\_releases.schedulable}\longrightarrow \mathbf{Skip}\right)$$

$$
\begin{array}{l}
\mathit{ReleaseLong} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\mathit{releaseLong.schedulable}\,?\,\mathit{longData} \longrightarrow \\
\mathit{data} := \mathit{longData}; \\
\mathit{pending} := \mathbf{True}; \\
\mathit{ReleaseLong}
\end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
\mathit{handleAsyncLongEventRet.schedulable} \longrightarrow \\
\mathbf{if}\ \mathit{pending} = \mathbf{True} \longrightarrow \\
\qquad \left(\begin{array}{l}
\mathit{pending} := \mathbf{False}; \\
\mathit{release\_complete}\,.\,\mathit{schedulable} \longrightarrow \\
\mathit{handleAsyncLongEventCall.schedulable.data} \longrightarrow \\
\mathit{ReleaseLong}
\end{array}\right) \\
[\!]\ \mathit{pending} = \mathbf{False} \longrightarrow \\
\qquad \mathit{ReadyLong} \\
\mathbf{fi}
\end{array}\right) \\
\quad \Box \\
\quad \left(\mathit{end\_releases.schedulable} \longrightarrow \mathbf{Skip}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{Cleanup} \mathrel{\widehat=} \\
\quad \mathit{cleanupSchedulableCall}\,.\,\mathit{schedulable} \longrightarrow \\
\quad \mathit{cleanupSchedulableRet}\,.\,\mathit{schedulable} \longrightarrow \mathbf{Skip}
\end{array}
$$

$$
\bullet \left(\mu X \bullet \left(\begin{array}{l}
\mathit{APEH\_Init}\ ;\ \mathit{Start}; \\
\left(\begin{array}{l}
\mathbf{if}\ \mathit{applicationTerminating} = \mathbf{False} \longrightarrow \\
\qquad \left(\mathit{Execute}\ ;\ \mathit{Cleanup}\ ;\ X\right) \\
[\!]\ \mathit{applicationTerminating} = \mathbf{True} \longrightarrow \\
\qquad \left(\mathit{end\_aperiodic\_app}\,.\,\mathit{schedulable} \longrightarrow \mathbf{Skip}\right) \\
\mathbf{fi}
\end{array}\right)
\end{array}\right)\right)
$$

end

C.14 OneShotEventHandlerFW

section $OneShotEventHandlerFW$ **parents** $MissionChan, SchedulableChan,$
$SchedulableId, MissionId, MissionIds, TopLevelMissionSequencerChan,$
$OneShotEventHandlerChan, SafeletMethChan, FrameworkChan,$
$AperiodicParameters$

process $OneShotEventHandlerFW \mathrel{\widehat=}$
$schedulable : SchedulableID;\ startTime : JTime;$
$aperiodicParameters : AperiodicParameters \bullet$ **begin**

state $OSEH\_State$

$controllingMission : MissionID$
$applicationTerminating : \mathbb{B}$
$deadline : JTime$
$deadlineMissHandler : SchedulableID$

state $OSEH\_State$

$OSEH\_Init$

$OSEH\_State'$

$controllingMission' = nullMissionId$
$applicationTerminating' = \mathbf{False}$
$deadline' = deadlineOfAperiodic(aperiodicParameters)$
$deadlineMissHandler' = missHandlerOfAperiodic(aperiodicParameters)$

$$Start \mathrel{\widehat=} \begin{pmatrix} Register; \\ Activate \end{pmatrix} \Box \begin{pmatrix} activate\_schedulables?someMissionID \longrightarrow \\ Start \end{pmatrix} \Box \begin{pmatrix} done\_toplevel\_sequencer \longrightarrow \\ applicationTerminating := \mathbf{True} \end{pmatrix}$$

$Register \mathrel{\widehat=}$
$register\,.\,schedulable\,?\,mID \longrightarrow$
$controllingMission := mID$

$Activate \mathrel{\widehat=}$
$activate\_schedulables\,.\,controllingMission \longrightarrow$
Skip

$$
\begin{array}{l}
Execute \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
Run \quad \llbracket \varnothing \mid MethodsSync \mid \varnothing \rrbracket \\
Methods
\end{array}\right) \\
\llbracket \varnothing \mid \{\!|\, end\_releases \,|\!\} \mid \varnothing \rrbracket \\
SignalTermination
\end{array}\right) \\
\quad \llbracket \varnothing \mid STCSync \mid \{ startTime \} \rrbracket \\
StartTimeController
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
Run \mathrel{\widehat=} \\
\quad \mathbf{if}\ deadlineMissHandler = nullSchedulableId \longrightarrow \\
\qquad \left(\begin{array}{l}
ScheduleOrWait \\
\llbracket \varnothing \mid ReleaseSync \mid \varnothing \rrbracket \\
Release
\end{array}\right) \\
\quad [\!]\ deadlineMissHandler \neq nullSchedulableId \longrightarrow \\
\qquad \left(\begin{array}{l}
\left(\begin{array}{l}
ScheduleOrWait \\
\llbracket \varnothing \mid ReleaseSync \mid \varnothing \rrbracket \\
Release
\end{array}\right) \\
\llbracket \varnothing \mid DeadlineSync \mid \varnothing \rrbracket \\
DeadlineClock
\end{array}\right) \\
\quad \mathbf{fi}
\end{array}
$$

$$
\begin{array}{l}
ScheduleOrWait \mathrel{\widehat=} \\
\quad get\_startTime\,.\,schedulable\,?\,startTime \longrightarrow \\
\quad \mathbf{if}\ startTime! = NULL \longrightarrow \\
\qquad Scheduled \\
\quad [\!]\ startTime = NULL \longrightarrow \\
\qquad NotScheduled \\
\quad \mathbf{fi}
\end{array}
$$

$$
\begin{array}{l}
Release \mathrel{\widehat=} \\
\quad \left(\begin{array}{l}
handleAsyncEventCall\,.\,schedulable \longrightarrow \\
handleAsyncEventRet\,.\,schedulable \longrightarrow \\
release\_complete\,.\,schedulable \longrightarrow \\
Release
\end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
reschedule\_handler\,.\,schedulable\,?\,newStartTime \longrightarrow \\
set\_startTime\,.\,schedulable\,!\,newStartTime \longrightarrow \\
Release
\end{array}\right) \\
\quad \Box \\
\quad \left(\begin{array}{l}
end\_releases\,.\,schedulable \longrightarrow \\
stop\_release\,.\,schedulable \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{DeadlineClock} \mathrel{\widehat=} \\
\quad \mathit{release} \,.\, \mathit{schedulable} \longrightarrow \\
\quad \left(\left(\begin{array}{l}
\left(\begin{array}{l} \mathbf{wait}\ \mathit{valueOf}(\mathit{deadline}); \\ \mathit{release} \,.\, \mathit{deadlineMissHandler} \longrightarrow \\ \mathit{DeadlineClock} \end{array} \right) \\
\Box \\
\left(\begin{array}{l} \mathit{release\_complete} \,.\, \mathit{schedulable} \longrightarrow \\ \mathit{DeadlineClock} \end{array} \right) \\
\Box \\
\left(\begin{array}{l} \mathit{deschedule\_handler} \,.\, \mathit{schedulable} \longrightarrow \\ \mathit{DeadlineClock} \end{array} \right)
\end{array} \right) \right) \\
\quad \triangle \left(\begin{array}{l} \mathit{end\_releases} \,.\, \mathit{schedulable} \longrightarrow \\ \mathbf{Skip} \end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{Scheduled} \mathrel{\widehat=} \\
\quad \mathit{get\_startTime} \,.\, \mathit{schedulable} \,?\, \mathit{startTime} \longrightarrow \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l} \mathbf{wait}\ \mathit{valueOf}(\mathit{startTime}) \\ \mathit{release} \,.\, \mathit{schedulable} \longrightarrow \\ \mathit{handleAsyncEventCall} \,.\, \mathit{schedulable} \longrightarrow \\ \mathit{NotScheduled} \end{array} \right) \\
\triangle \\
\left(\begin{array}{l}
\left(\begin{array}{l} \mathit{deschedule\_handler} \,.\, \mathit{schedulable} \longrightarrow \\ \mathit{NotScheduled} \end{array} \right) \\
\Box \\
\left(\begin{array}{l} \mathit{reschedule\_handler} \,.\, \mathit{schedulable} \,?\, \mathit{newStartTime} \longrightarrow \\ \mathit{set\_startTime} \,.\, \mathit{schedulable} \,!\, \mathit{newStartTime} \longrightarrow \\ \mathit{Scheduled} \end{array} \right)
\end{array} \right)
\end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
\mathit{NotScheduled} \mathrel{\widehat=} \\
\quad \left(\begin{array}{l} \mathit{deschedule\_handler} \,.\, \mathit{schedulable} \longrightarrow \\ \mathit{NotScheduled} \end{array} \right) \\
\quad \Box \\
\quad \left(\begin{array}{l} \mathit{reschedule\_handler} \,.\, \mathit{schedulable} \,?\, \mathit{newStartTime} \longrightarrow \\ \mathit{set\_startTime} \,.\, \mathit{schedulable} \,!\, \mathit{newStartTime} \longrightarrow \\ \mathit{Scheduled} \end{array} \right) \\
\quad \Box \\
\quad \left(\begin{array}{l} \mathit{end\_releases} \,.\, \mathit{schedulable} \longrightarrow \\ \mathbf{Skip} \end{array} \right)
\end{array}
$$

```
$Methods \mathrel{\widehat=}$
    $\left(\begin{array}{l} Deschedule; \\ Methods \end{array}\right)$
    $\Box$
    $\left(\begin{array}{l} GetNextReleaseTime; \\ Methods \end{array}\right)$
    $\Box$
    $\left(\begin{array}{l} ScheduleNextRelease; \\ Methods \end{array}\right)$
    $\Box$
    $\left(\begin{array}{l} end\_releases~.~schedulable \longrightarrow \\ \mathbf{Skip} \end{array}\right)$

$Deschedule \mathrel{\widehat=}$
    $\mathbf{var}~wasScheduled : \mathbb{B} \bullet$
    $descheduleCall~.~schedulable \longrightarrow$
    $deschedule\_handler.schedulable \longrightarrow$
    $get\_startTime~.~schedulable~?~startTime \longrightarrow$
    (if $startTime = NULL \longrightarrow$
            $wasScheduled := \mathbf{False}$
     $[\!]~startTime \neq NULL \longrightarrow$
            $wasScheduled := \mathbf{True}$
     fi ;
     $set\_startTime~.~schedulable~!~NULL \longrightarrow$
     $descheduleRet~.~schedulable~!~wasScheduled \longrightarrow$
     Skip)

$GetNextReleaseTime \mathrel{\widehat=}$
    $getNextReleaseTimeCall~.~schedulable \longrightarrow$
    $get\_startTime~.~schedulable~?~startTime \longrightarrow$
    $getNextReleaseTimeRet~.~schedulable~!~startTime \longrightarrow$
    Skip

$ScheduleNextRelease \mathrel{\widehat=}$
    $scheduleNextRelease~.~schedulable~?~newStartTime \longrightarrow$
    $set\_startTime~.~schedulable~!~newStartTime \longrightarrow$
    if $newStartTime = NULL \longrightarrow$
        $\left(\begin{array}{l} deschedule\_handler.schedulable \longrightarrow \\ \mathbf{Skip} \end{array}\right)$
    $[\!]~newStartTime \neq NULL \longrightarrow$
        $\left(\begin{array}{l} reschedule\_handler~!~schedulable~!~newStartTime \longrightarrow \\ \mathbf{Skip} \end{array}\right)$
    fi
```

$SignalTermination \mathrel{\widehat=}$
$\quad signalTerminationCall\,.\,schedulable \longrightarrow$
$\quad end\_releases\,.\,schedulable \longrightarrow$
$\quad signalTerminationRet\,.\,schedulable \longrightarrow$
$\quad done\_schedulable\,.\,schedulable \longrightarrow$
$\quad \mathbf{Skip}$

$$StartTimeController \mathrel{\widehat=} \left(\begin{array}{l} get\_startTime\,.\,schedulable\,!\,startTime \longrightarrow \\ StartTimeController \end{array}\right) \;\Box\; \left(\begin{array}{l} set\_startTime\,.\,schedulable\,?\,newStartTime \longrightarrow \\ StartTimeController \end{array}\right)$$

$Cleanup \mathrel{\widehat=}$
$\quad cleanupSchedulableCall\,.\,schedulable \longrightarrow$
$\quad cleanupSchedulableRet\,.\,schedulable \longrightarrow$
$\mathbf{Skip}$

$$\bullet \left(\mu X \bullet \left(\begin{array}{l} OSEH\_Init\,;\; Start; \\ \left(\begin{array}{l} \mathbf{if}\ applicationTerminating = \mathbf{False} \longrightarrow \\ \quad \left(Execute\,;\; Cleanup\,;\; X \right) \\ [\!]\ applicationTerminating = \mathbf{True} \longrightarrow \\ \quad \left(\begin{array}{l} end\_oneShot\_app\,.\,schedulable \longrightarrow \\ \mathbf{Skip} \end{array} \right) \\ \mathbf{fi} \end{array} \right) \end{array} \right) \right)$$

end

C.15 ManagedThreadFW

section *ManagedThreadFW* **parents** *ManagedThreadChan*, *SchedulableId*, *MissionId*, *MissionIds*, *TopLevelMissionSequencerChan*, *SchedulableChan*, *SafeletMethChan*, *FrameworkChan*

process $ManagedThreadFW \mathrel{\widehat=} schedulable : SchedulableID \bullet$ **begin**

state *MT_State*

$controllingMission : MissionID$
$applicationTerminating : \mathbb{B}$

state *MT_State*

MT_Init

MT_State'

$controllingMission' = nullMissionId$
$applicationTerminating' = \mathbf{False}$

$$Start \mathrel{\widehat=} \begin{pmatrix} Register; \\ Activate \end{pmatrix} \;\Box\; \begin{pmatrix} activate\_schedulables?someMissionID \longrightarrow \\ Start \end{pmatrix} \;\Box\; \begin{pmatrix} done\_toplevel\_sequencer \longrightarrow \\ applicationTerminating := \mathbf{True} \end{pmatrix}$$

$Register \mathrel{\widehat=}$
$register\,.\,schedulable\,?\,mID \longrightarrow$
$controllingMission := mID$

$Activate \mathrel{\widehat=}$
$activate\_schedulables\,.\,controllingMission \longrightarrow$
$\mathbf{Skip}$

$Execute \mathrel{\widehat=} Run \llbracket \varnothing \mid \{\!|\, done\_schedulable \,|\!\} \mid \varnothing \rrbracket Methods$

$Run \mathrel{\widehat=}$
$\quad runCall \,.\, schedulable \longrightarrow$
$\quad runRet \,.\, schedulable \longrightarrow$
$\quad done\_schedulable \,.\, schedulable \longrightarrow$
$\quad$ **Skip**

$Methods \mathrel{\widehat=}$
$\quad \left(SignalTerminationMeth \;;\; Methods \right)$
$\quad \Box$
$\quad done\_schedulable \,.\, schedulable \longrightarrow$
$\quad$ **Skip**

$SignalTerminationMeth \mathrel{\widehat=}$
$\quad signalTerminationCall \,.\, schedulable \longrightarrow$
$\quad signalTerminationRet \,.\, schedulable \longrightarrow$
$\quad$ **Skip**

$Cleanup \mathrel{\widehat=}$
$\quad cleanupSchedulableCall \,.\, schedulable \longrightarrow$
$\quad cleanupSchedulableRet \,.\, schedulable \longrightarrow$
$\quad$ **Skip**

$$\bullet \left(\begin{array}{l} \mu X \bullet MT\_Init \;;\; Start; \\ \left(\begin{array}{l} \mathbf{if}\ applicationTerminating = \mathbf{False} \longrightarrow \\ \qquad \left(Execute \;;\; Cleanup \;;\; X \right) \\ [\!]\ applicationTerminating = \mathbf{True} \longrightarrow \\ \qquad \left(\begin{array}{l} end\_managedThread\_app \,.\, schedulable \longrightarrow \\ \mathbf{Skip} \end{array} \right) \\ \mathbf{fi} \end{array} \right) \end{array} \right)$$

end

Appendix D

Circus BNF Encoding

This appendix presents our encoding in Z of the BNF description of *Circus*.

$CircusProgram == \mathrm{seq}\, CircusParagraph$

$CircusParagraph ::= para\langle\langle Paragraph\rangle\rangle \mid chanDef\langle\langle ChannelDefinition\rangle\rangle \mid$
$\quad chanSetDef\langle\langle ChanSetDefinition\rangle\rangle \mid procDef\langle\langle ProcDefinition\rangle\rangle$

$ChannelDefinition == CDeclaration$

$CDeclaration ::= scDecl\langle\langle SCDeclaration\rangle\rangle \mid multiDecl\langle\langle SCDeclaration \times CDeclaration\rangle\rangle$

$SCDeclaration ::= chanName\langle\langle \mathrm{seq}\, N\rangle\rangle \mid chanNameWithType\langle\langle \mathrm{seq}\, N \times Expression\rangle\rangle \mid$
$\quad scSe\langle\langle SchemaExp\rangle\rangle$

$ChanSetDefinition ::= csdName\langle\langle N \times CSExpression\rangle\rangle$

$ProcDefinition ::= pd\langle\langle N \times Process\rangle\rangle$

$Process ::= proc\langle\langle \mathrm{seq}\, PParagraph \times Action\rangle\rangle \mid procName\langle\langle N\rangle\rangle \mid$
$\quad procSeq\langle\langle Process \times Process\rangle\rangle \mid procExtChoice\langle\langle Process \times Process\rangle\rangle \mid$
$\quad procIntChoice\langle\langle Process \times Process\rangle\rangle \mid procPar\langle\langle Process \times CSExpression \times Process\rangle\rangle \mid$
$\quad procInter\langle\langle Process \times Process\rangle\rangle \mid procHide\langle\langle Process \times CSExpression\rangle\rangle \mid$
$\quad procRename\langle\langle Process \times \mathrm{seq}\, N \times \mathrm{seq}\, N\rangle\rangle \mid procParam\langle\langle Declaration \times Process\rangle\rangle \mid$
$\quad procInstP\langle\langle Process \times \mathrm{seq}\, Expression\rangle\rangle \mid procGeneric\langle\langle \mathrm{seq}\, N \times Process\rangle\rangle \mid$
$\quad procInstG\langle\langle Process \times \mathrm{seq}\, Expression\rangle\rangle \mid$
$\quad procItrInter\langle\langle Declaration \times Process\rangle\rangle$

$PParagraph ::= pPar\langle\langle Paragraph\rangle\rangle \mid def\langle\langle N \times Action\rangle\rangle$

$GuardedAction ::= thenAct\langle\langle Predicate \times Action\rangle\rangle \mid$
$\quad thenActComp\langle\langle Predicate \times Action \times GuardedAction\rangle\rangle$

$Action ::= actSe\langle\!\langle SchemaExp\rangle\!\rangle \mid com\langle\!\langle Command\rangle\!\rangle \mid skip \mid stop \mid chaos \mid$
$\quad prefixExp\langle\!\langle Communication \times Action\rangle\!\rangle \mid$
$\quad guard\langle\!\langle Predicate \times Action\rangle\!\rangle \mid seqExp\langle\!\langle Action \times Action\rangle\!\rangle \mid$
$\quad extChoice\langle\!\langle Action \times Action\rangle\!\rangle \mid intChoice\langle\!\langle Action \times Action\rangle\!\rangle \mid$
$\quad actPar\langle\!\langle Action \times CSExpression \times Action\rangle\!\rangle \mid actInter\langle\!\langle Action \times Action\rangle\!\rangle \mid$
$\quad actHide\langle\!\langle Action \times CSExpression\rangle\!\rangle \mid mu\langle\!\langle N \times Action\rangle\!\rangle \mid actParam\langle\!\langle Declaration \times Action\rangle\!\rangle \mid$
$\quad actInst\langle\!\langle Action \times \mathrm{seq}\, Expression\rangle\!\rangle$

$CSExpression ::= cs\langle\!\langle \mathrm{seq}\, N\rangle\!\rangle \mid csName\langle\!\langle N\rangle\!\rangle \mid$
$\quad union\langle\!\langle CSExpression \times CSExpression\rangle\!\rangle \mid$
$\quad intersect\langle\!\langle CSExpression \times CSExpression\rangle\!\rangle \mid$
$\quad subtract\langle\!\langle CSExpression \times CSExpression\rangle\!\rangle$

$Communication == N \times \mathrm{seq}\, CParameter$

$CParameter ::= shriek\langle\!\langle N\rangle\!\rangle \mid shriekRestrict\langle\!\langle N \times Predicate\rangle\!\rangle \mid bang\langle\!\langle Expression\rangle\!\rangle \mid$
$\quad dotParam\langle\!\langle Expression\rangle\!\rangle$

$Command ::= spec\langle\!\langle \mathrm{seq}\, N \times Predicate \times Predicate\rangle\!\rangle \mid equals\langle\!\langle N \times \mathrm{seq}\, Expression\rangle\!\rangle$

$[Predicate, N, Expression, Paragraph, SchemaExp, Declaration]$

Appendix E

Formal Translation Functions

This appendix presents all of the translation functions discussed in Sect. 5.2. These functions are written in Z and translate SCJ programs into our *Circus* model. As discussed in Sect. 5.2, these functions formalise the core of our translation strategy.

Framework

This section presents the definition of the framework model. Since the *Circus* model of the framework is generic and reused, it is defined here as a *Circus* program composed of the model presented in Sect. C.

$\mathbf{section}\ Framework\ \mathbf{parents}\ scj\_prelude, SCJBNFEncoding, CircusBNFEncoding$

$[ID]$
$[Type]$

$$NullType : Type$$

$$\begin{array}{l} SafeletFWName : N \\ TopLevelMissionSequencerFWNMame : N \end{array}$$

$$\begin{array}{l} controlTierSync : CSExpression \\ Tier0 : N \\ MissionIds : \operatorname{seq} CircusParagraph \\ SchedulableIds : \operatorname{seq} CircusParagraph \\ ThreadIds : \operatorname{seq} CircusParagraph \\ ObjectIds : \operatorname{seq} CircusParagraph \end{array}$$

$$\begin{array}{l} ServicesChan : \operatorname{seq} CircusParagraph \\ GlobalTypes : \operatorname{seq} CircusParagraph \\ JTime : \operatorname{seq} CircusParagraph \end{array}$$

$PrimitiveTypes : \mathrm{seq}\ CircusParagraph$
$Priority : \mathrm{seq}\ CircusParagraph$
$PriorityQueue : \mathrm{seq}\ CircusParagraph$

$FrameworkChan : \mathrm{seq}\ CircusParagraph$
$MissionId : \mathrm{seq}\ CircusParagraph$
$SchedulableId : \mathrm{seq}\ CircusParagraph$

$ObjectFW : CircusParagraph$
$ObjectChan : \mathrm{seq}\ CircusParagraph$
$ObjectFWChan : \mathrm{seq}\ CircusParagraph$
$ObjectMethChan : \mathrm{seq}\ CircusParagraph$
$ThreadFW : CircusParagraph$
$ThreadChan : \mathrm{seq}\ CircusParagraph$
$ThreadFWChan : \mathrm{seq}\ CircusParagraph$
$ThreadMethChan : \mathrm{seq}\ CircusParagraph$

$SafeletFW : CircusParagraph$
$SafeletFWChan : \mathrm{seq}\ CircusParagraph$
$SafeletChan : \mathrm{seq}\ CircusParagraph$
$SafeletMethChan : \mathrm{seq}\ CircusParagraph$

$TopLevelMissionSequencerFW : CircusParagraph$
$TopLevelMissionSequencerChan : \mathrm{seq}\ CircusParagraph$
$TopLevelMissionSequencerFWChan : \mathrm{seq}\ CircusParagraph$

$MissionSequencerChan : \mathrm{seq}\ CircusParagraph$
$MissionSequencerFWChan : \mathrm{seq}\ CircusParagraph$
$MissionSequencerMethChan : \mathrm{seq}\ CircusParagraph$

$MissionFW : CircusParagraph$
$MissionChan : \mathrm{seq}\ CircusParagraph$
$MissionFWChan : \mathrm{seq}\ CircusParagraph$
$MissionMethChan : \mathrm{seq}\ CircusParagraph$

$$\begin{array}{|l}
SchedulableChan : \mathrm{seq}\ CircusParagraph \\
SchedulableMethChan : \mathrm{seq}\ CircusParagraph \\
SchedulableFWChan : \mathrm{seq}\ CircusParagraph \\
HandlerChan : \mathrm{seq}\ CircusParagraph \\
HandlerFWChan : \mathrm{seq}\ CircusParagraph \\
HandlerMethChan : \mathrm{seq}\ CircusParagraph
\end{array}$$

$$\begin{array}{|l}
PeriodicEventHandlerChan : \mathrm{seq}\ CircusParagraph \\
PeriodicEventHandlerFW : CircusParagraph \\
PeriodicEventHandlerFWChan : \mathrm{seq}\ CircusParagraph \\
PeriodicParameters : \mathrm{seq}\ CircusParagraph
\end{array}$$

$$\begin{array}{|l}
AperiodicEventHandlerChan : \mathrm{seq}\ CircusParagraph \\
AperiodicEventHandlerFW : CircusParagraph \\
AperiodicLongEventHandlerMethChan : \mathrm{seq}\ CircusParagraph \\
AperiodicParameters : \mathrm{seq}\ CircusParagraph
\end{array}$$

$$\begin{array}{|l}
OneShotEventHandlerChan : \mathrm{seq}\ CircusParagraph \\
OneShotEventHandlerFW : CircusParagraph \\
OneShotEventHandlerFWChan : \mathrm{seq}\ CircusParagraph \\
OneShotEventHandlerMethChan : \mathrm{seq}\ CircusParagraph
\end{array}$$

$$\begin{array}{|l}
SchedulableMissionSequencerFW : CircusParagraph \\
SchedulableMissionSequencerChan : \mathrm{seq}\ CircusParagraph \\
SchedulableMissionSequencerFWChan : \mathrm{seq}\ CircusParagraph
\end{array}$$

$$\begin{array}{|l}
ManagedThreadFW : CircusParagraph \\
ManagedThreadChan : \mathrm{seq}\ CircusParagraph \\
ManagedThreadFWChan : \mathrm{seq}\ CircusParagraph \\
ManagedThreadMethChan : \mathrm{seq}\ CircusParagraph
\end{array}$$

$$
\begin{array}{l}
framework : CircusProgram \\
\hline
framework = ServicesChan \frown GlobalTypes \frown JTime \frown PrimitiveTypes \frown \\
\quad Priority \frown PriorityQueue \frown FrameworkChan \frown MissionId \frown \\
\quad SchedulableId \frown \langle ObjectFW \rangle \frown ObjectChan \frown \\
\quad ObjectFWChan \frown ObjectMethChan \frown \langle ThreadFW \rangle \frown \\
\quad ThreadChan \frown ThreadFWChan \frown ThreadMethChan \frown \langle SafeletFW \rangle \frown \\
\quad SafeletFWChan \frown SafeletChan \frown SafeletMethChan \frown \\
\quad \langle TopLevelMissionSequencerFW \rangle \frown TopLevelMissionSequencerChan \frown \\
\quad TopLevelMissionSequencerFWChan \frown MissionSequencerChan \frown \\
\quad MissionSequencerFWChan \frown MissionSequencerMethChan \frown \langle MissionFW \rangle \frown \\
\quad MissionChan \frown MissionFWChan \frown MissionMethChan \frown SchedulableChan \frown \\
\quad SchedulableMethChan \frown SchedulableFWChan \frown HandlerChan \frown HandlerFWChan \frown \\
\quad HandlerMethChan \frown \langle PeriodicEventHandlerFW \rangle \frown \\
\quad PeriodicEventHandlerChan \frown PeriodicEventHandlerFWChan \frown PeriodicParameters \frown \\
\quad AperiodicEventHandlerChan \frown \langle AperiodicEventHandlerFW \rangle \frown \\
\quad AperiodicLongEventHandlerMethChan \frown AperiodicParameters \frown \\
\quad OneShotEventHandlerChan \frown \langle OneShotEventHandlerFW \rangle \frown \\
\quad OneShotEventHandlerFWChan \frown OneShotEventHandlerMethChan \frown \\
\quad \langle SchedulableMissionSequencerFW \rangle \frown \\
\quad SchedulableMissionSequencerChan \frown SchedulableMissionSequencerFWChan \frown \\
\quad \langle ManagedThreadFW \rangle \frown ManagedThreadChan \frown \\
\quad ManagedThreadFWChan \frown ManagedThreadMethChan
\end{array}
$$

Build Phase

This section presents the functions that constitute the build phase. This phase extracts the application-specific information from an SCJ program and build an environment for each component.

section *BuildPhase* **parents** *scj_prelude*, *SCJBNFEncoding*, *CircusBNFEncoding*, *Framework*

$$
\begin{array}{l}
TranslatablePrograms : \mathbb{P}\, SCJProgram \\
\hline
TranslatablePrograms = \\
\quad \{ s : SCJProgram \mid \\
\quad\quad ProgTLMS(s) \neq NoSequencer \\
\quad\quad \land ProgTiers(s) \neq \langle\rangle \\
\quad\quad \land ProgClusters(s) \neq \varnothing \\
\quad\quad \land \forall c : ProgClusters(s) \\
\quad\quad\quad \bullet ClusterSchedulables(c) \neq \varnothing \}
\end{array}
$$

$AppEnv$

$Name : N$

$Parameters : \mathrm{seq}\, Expression$

$ClusterAppEnv$

$Mission : AppEnv$

$Schedulables : \mathbb{F}\, AppEnv$

$Schedulables \neq \varnothing$

$TierAppEnv$

$Clusters : \mathrm{seq}\, ClusterAppEnv$

$Clusters \neq \langle\rangle$

$AppProcEnv$

$Safelet : AppEnv$

$TopLevelMS : AppEnv$

$Tiers : \mathrm{seq}\, TierAppEnv$

$Tiers \neq \langle\rangle$

$GetSafeletAppEnv : AppProcEnv \rightarrow AppEnv$

$$\forall a : AppProcEnv \bullet \\ \quad GetSafeletAppEnv(a) = a.Safelet$$

$GetTLMSAppEnv : AppProcEnv \rightarrow AppEnv$

$$\forall a : AppProcEnv \bullet \\ \quad GetTLMSAppEnv(a) = a.TopLevelMS$$

$GetTiersAppEnv : AppProcEnv \rightarrow \mathrm{seq}\, TierAppEnv$

$$\forall a : AppProcEnv \bullet \\ \quad GetTiersAppEnv(a) = a.Tiers$$

$IDof : Identifier \rightarrow N$

$ParamsOf : \text{seq}\, ClassBodyDeclaration \rightarrow \text{seq}\, Expression$

$BuildSOAppEnv : \mathbb{P}\, SchedulableObject \rightarrow \mathbb{F}\, AppEnv$

$$
\begin{array}{l}
\forall\, scheds : \mathbb{P}\, SchedulableObject \\
\quad \bullet \exists\, manT : ManagedThread;\ nestMS : NestedMissionSequencer;\ eh : EventHandler \\
\qquad perEH : PeriodicEventHandler;\ oneEH : OneShotEventHandler; \\
\qquad apehShort : Identifier \times EventHandlerClassBody; \\
\qquad apehLong : Identifier \times LongEventHandlerClassBody \\
\qquad \bullet BuildSOAppEnv(scheds) = \{a : AppEnv \\
\qquad\quad \mid \forall\, so : scheds \bullet \exists\, name : N;\ params : \text{seq}\, Expression \\
\qquad\qquad \mid so = mt(manT) \Rightarrow \\
\qquad\qquad\quad name = IDof(manT.1) \land params = ParamsOf(manT.2.2) \\
\qquad\qquad \land\, so = nms(nestMS) \Rightarrow \\
\qquad\qquad\quad name = IDof(nestMS.1) \land params = ParamsOf(nestMS.2.2) \\
\qquad\qquad \land\, so = handler(pehDecl(perEH)) \Rightarrow \\
\qquad\qquad\quad name = IDof(perEH.1) \land params = ParamsOf(perEH.2.2) \\
\qquad\qquad \land\, so = handler(osehDecl(oneEH)) \Rightarrow \\
\qquad\qquad\quad name = IDof(oneEH.1) \land params = ParamsOf(oneEH.2.2) \\
\qquad\qquad \land\, so = handler(apehDecl(apehType(apehShort))) \Rightarrow \\
\qquad\qquad\quad name = IDof(apehShort.1) \land params = ParamsOf(apehShort.2.2) \\
\qquad\qquad \land\, so = handler(apehDecl(aplehType(apehLong))) \Rightarrow \\
\qquad\qquad\quad name = IDof(apehLong.1) \land params = ParamsOf(apehLong.2.2) \\
\qquad \bullet a = \langle\!| Name == name, Parameters == params |\!\rangle\}
\end{array}
$$

$BuildClusterAppEnv : Cluster \rightarrow ClusterAppEnv$

$$
\begin{array}{l}
\forall\, c : Cluster \\
\quad \bullet \exists\, m : Mission;\ seqSO : \mathbb{F}\, SchedulableObject \\
\qquad \mid c = (m, seqSO) \\
\qquad \bullet BuildClusterAppEnv(c) = \\
\qquad\quad \langle\!| Mission == \langle\!| Name == IDof(m.1), Parameters == ParamsOf(m.2.3) |\!\rangle, \\
\qquad\qquad Schedulables == BuildSOAppEnv(seqSO) |\!\rangle
\end{array}
$$

$BuildClusterAppEnvs : \text{seq}\, Cluster \rightarrow \text{seq}\, ClusterAppEnv$

$$
\begin{array}{l}
\mathit{BuildTierAppEnv} : \mathit{Tier} \rightarrow \mathit{TierAppEnv} \\
\hline
\forall \mathit{tier} : \mathit{Tier} \\
\quad \bullet \mathit{BuildTierAppEnv}(\mathit{tier}) = \lparen\!| \mathit{Clusters} == \mathit{BuildClusterAppEnvs}(\mathit{tier}) |\!\rparen
\end{array}
$$

$$
\begin{array}{l}
\mathit{BuildTiersAppEnv} : \mathrm{seq}\ \mathit{Tier} \rightarrow \mathrm{seq}\ \mathit{TierAppEnv} \\
\hline
\forall \mathit{tiers} : \mathrm{seq}\ \mathit{Tier} \\
\quad \bullet \# \mathit{tiers} = 1 \Rightarrow \mathit{BuildTiersAppEnv}(\mathit{tiers}) = \langle \mathit{BuildTierAppEnv}(\mathit{head}\ \mathit{tiers}) \rangle \\
\quad \land \# \mathit{tiers} \geq 1 \Rightarrow \mathit{BuildTiersAppEnv}(\mathit{tiers}) = \\
\qquad \langle \mathit{BuildTierAppEnv}(\mathit{head}\ \mathit{tiers}) \rangle \frown \mathit{BuildTiersAppEnv}(\mathit{tail}\ \mathit{tiers})
\end{array}
$$

$$
\begin{array}{l}
\mathit{BuildAppProcEnv} : \mathit{SCJProgram} \nrightarrow \mathit{AppProcEnv} \\
\hline
\mathit{dom}\ \mathit{BuildAppProcEnv} = \mathit{TranslatablePrograms} \\
\forall \mathit{scjProg} : \mathit{dom}\ \mathit{BuildAppProcEnv} \\
\quad \bullet \exists \mathit{safelet} : \mathit{Safelet};\ \mathit{tiers} : \mathrm{seq}\ \mathit{Tier};\ \mathit{ms} : \mathit{MissionSequencer} \\
\qquad | \ \mathit{safelet} = \mathit{ProgSafelet}(\mathit{scjProg}) \\
\qquad \land \mathit{tlms}(\mathit{ms}) = \mathit{ProgTLMS}(\mathit{scjProg}) \\
\qquad \land \mathit{tiers} = \mathit{ProgTiers}(\mathit{scjProg}) \\
\qquad \quad \bullet \exists \mathit{sfEnv} : \mathit{AppEnv};\ \mathit{tlmsEnv} : \mathit{AppEnv}; \\
\qquad \qquad \mathit{tiersEnv} : \mathrm{seq}\ \mathit{TierAppEnv} \\
\qquad \qquad \quad \bullet \mathit{sfEnv} = \lparen\!| \mathit{Name} == \mathit{IDof}(\mathit{safelet}.1), \\
\qquad \qquad \qquad \mathit{Parameters} == \mathit{ParamsOf}(\mathit{safelet}.2.4) |\!\rparen \\
\qquad \qquad \quad \land \mathit{tlmsEnv} = \lparen\!| \mathit{Name} == \mathit{IDof}(\mathit{ms}.1), \\
\qquad \qquad \qquad \mathit{Parameters} == \mathit{ParamsOf}(\mathit{ms}.2.2) |\!\rparen \\
\qquad \qquad \quad \land \mathit{tiersEnv} = \mathit{BuildTiersAppEnv}(\mathit{tiers}) \\
\qquad \qquad \quad \land \mathit{BuildAppProcEnv}(\mathit{scjProg}) = \lparen\!| \mathit{Safelet} == \mathit{sfEnv}, \\
\qquad \qquad \qquad \mathit{TopLevelMS} == \mathit{tlmsEnv}, \mathit{Tiers} == \mathit{tiersEnv} |\!\rparen
\end{array}
$$

ClusterEnv

$$
\begin{array}{l}
\mathit{Mission} : \mathit{Identifier} \\
\mathit{NestedMissionSequencers} : \mathbb{P}\ \mathit{Identifier} \\
\mathit{ManagedThreads} : \mathbb{P}\ \mathit{Identifier} \\
\mathit{PeriodicEventHandlers} : \mathbb{P}\ \mathit{Identifier} \\
\mathit{AperiodicEventHandlers} : \mathbb{P}\ \mathit{Identifier} \\
\mathit{OneShotEventHandlers} : \mathbb{P}\ \mathit{Identifier} \\
\hline
\mathit{disjoint} \langle \mathit{NestedMissionSequencers}, \mathit{ManagedThreads}, \mathit{PeriodicEventHandlers}, \\
\quad \mathit{AperiodicEventHandlers}, \mathit{OneShotEventHandlers} \rangle \\
\quad \bigcup \{ \mathit{NestedMissionSequencers}, \mathit{ManagedThreads}, \mathit{PeriodicEventHandlers}, \\
\quad \mathit{AperiodicEventHandlers}, \mathit{OneShotEventHandlers} \} \neq \varnothing
\end{array}
$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{TierEnv}} \\
\hline
\mathit{Clusters} : \mathrm{seq}\ \mathit{ClusterEnv} \\
\hline
\mathit{Clusters} \neq \langle\rangle \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{FWEnv}} \\
\hline
\mathit{TopLevelMS} : \mathit{Identifier} \\
\mathit{Tiers} : \mathrm{seq}\ \mathit{TierEnv} \\
\hline
\mathit{Tiers} \neq \langle\rangle \\
\hline
\end{array}$$

$$\begin{array}{|l}
\mathit{GetTierFWEnvs} : \mathit{FWEnv} \rightarrow \mathrm{seq}\ \mathit{TierEnv} \\
\hline
\forall\, \mathit{env} : \mathit{FWEnv} \\
\quad \bullet\ \mathit{GetTierFWEnvs}(\mathit{env}) = \mathit{env}.\mathit{Tiers}
\end{array}$$

$$\begin{array}{|l}
\mathit{GetIdentifiers} : \mathbb{F}\, \mathit{SchedulableObject} \rightarrow \mathbb{F}\, \mathit{Identifier} \\
\hline
\forall\, \mathit{scheds} : \mathbb{F}\, \mathit{SchedulableObject} \\
\quad \bullet\ \exists\, \mathit{manT} : \mathit{ManagedThread};\ \mathit{nestMS} : \mathit{NestedMissionSequencer}; \\
\qquad \mathit{perEH} : \mathit{PeriodicEventHandler};\ \mathit{oneEH} : \mathit{OneShotEventHandler}; \\
\qquad \mathit{eh} : \mathit{EventHandler}; \\
\qquad \mathit{apehShort} : \mathit{Identifier} \times \mathit{EventHandlerClassBody}; \\
\qquad \mathit{apehLong} : \mathit{Identifier} \times \mathit{LongEventHandlerClassBody} \\
\qquad \bullet\ \mathit{GetIdentifiers}(\mathit{scheds}) = \{ i : \mathit{Identifier} \\
\qquad\quad \mid \forall\, s : \mathit{scheds} \\
\qquad\quad \bullet\ s = \mathit{mt}(\mathit{manT}) \Rightarrow i = \mathit{manT}.1 \\
\qquad\quad \land\ s = \mathit{nms}(\mathit{nestMS}) \Rightarrow i = \mathit{nestMS}.1 \\
\qquad\quad \land\ s = \mathit{handler}(\mathit{pehDecl}(\mathit{perEH})) \Rightarrow i = \mathit{perEH}.1 \\
\qquad\quad \land\ s = \mathit{handler}(\mathit{apehDecl}(\mathit{apehType}(\mathit{apehShort}))) \Rightarrow i = \mathit{apehShort}.1 \\
\qquad\quad \land\ s = \mathit{handler}(\mathit{apehDecl}(\mathit{aplehType}(\mathit{apehLong}))) \Rightarrow i = \mathit{apehLong}.1 \\
\qquad\quad \land\ s = \mathit{handler}(\mathit{osehDecl}(\mathit{oneEH})) \Rightarrow i = \mathit{oneEH}.1 \\
\qquad \}
\end{array}$$

$$
\begin{array}{|l}
BuildSOEnvs : \mathbb{F}\,SchedulableObject \rightarrow \\
\quad \mathbb{F}\,Identifier \times \mathbb{F}\,Identifier \times \mathbb{F}\,Identifier \times \\
\quad \mathbb{F}\,Identifier \times \mathbb{F}\,Identifier \\
\hline
\forall\, s : \mathbb{F}\,SchedulableObject \\
\quad \bullet \exists\, sms : \mathbb{F}\,Identifier;\ pehs : \mathbb{F}\,Identifier; \\
\qquad\quad apehs : \mathbb{F}\,Identifier;\ osehs : \mathbb{F}\,Identifier;\ mts : \mathbb{F}\,Identifier \\
\qquad \mid mts = GetIdentifiers(\{mtSched : s \\
\qquad\quad \mid \exists\, m : ManagedThread \\
\qquad\quad \bullet\ mtSched = mt(m)\}) \\
\qquad \wedge\ sms = GetIdentifiers(\{nmsSched : s \\
\qquad\quad \mid \exists\, n : NestedMissionSequencer \\
\qquad\quad \bullet\ nmsSched = nms(n)\}) \\
\qquad \wedge\ pehs = GetIdentifiers(\{pehSched : s \\
\qquad\quad \mid \exists\, p : PeriodicEventHandler \\
\qquad\quad \bullet\ pehSched = handler(pehDecl(p))\}) \\
\qquad \wedge\ apehs = GetIdentifiers(\{apehSched : s \\
\qquad\quad \mid \exists\, a : Identifier \times EventHandlerClassBody \\
\qquad\quad \bullet\ apehSched = handler(apehDecl(apehType(a)))\}) \\
\qquad \wedge\ apehs = GetIdentifiers(\{apehLSched : s \\
\qquad\quad \mid \exists\, a : Identifier \times LongEventHandlerClassBody \\
\qquad\quad \bullet\ apehLSched = handler(apehDecl(aplehType(a)))\}) \\
\qquad \wedge\ osehs = GetIdentifiers(\{osehSched : s \\
\qquad\quad \mid \exists\, o : OneShotEventHandler \\
\qquad\quad \bullet\ osehSched = handler(osehDecl(o))\}) \\
\quad \bullet\ BuildSOEnvs(s) = (sms, pehs, apehs, osehs, mts)
\end{array}
$$

$$
\begin{array}{|l}
BuildClusterEnv : Cluster \nrightarrow ClusterEnv \\
\hline
\forall\, c : Cluster \\
\quad \bullet \exists\, missionName : Identifier;\ sms : \mathbb{F}\,Identifier;\ pehs : \mathbb{F}\,Identifier; \\
\qquad apehs : \mathbb{F}\,Identifier;\ oseh : \mathbb{F}\,Identifier;\ mts : \mathbb{F}\,Identifier;\ cluster : ClusterEnv \\
\quad \mid missionName = c.1.1 \\
\quad \wedge\ (sms, pehs, apehs, oseh, mts) = BuildSOEnvs(c.2) \\
\quad \bullet\ BuildClusterEnv(c) = \\
\qquad \langle\!| Mission == missionName, NestedMissionSequencers == sms, \\
\qquad PeriodicEventHandlers == pehs, AperiodicEventHandlers == apehs, \\
\qquad OneShotEventHandlers == oseh, ManagedThreads == mts |\!\rangle
\end{array}
$$

$$
\begin{array}{|l}
\mathit{BuildClusterEnvs} : \operatorname{seq} \mathit{Cluster} \rightarrow \operatorname{seq} \mathit{ClusterEnv} \\
\hline
\forall c : \operatorname{seq} \mathit{Cluster} \\
\quad \mid c \neq \langle\rangle \land \forall s : \operatorname{seq} \mathit{Cluster} \bullet s \neq \langle\rangle \\
\quad \bullet \# c = 1 \Rightarrow \mathit{BuildClusterEnvs}(c) = \langle \mathit{BuildClusterEnv}(\mathit{head}\ c) \rangle \\
\quad \land \# c \geq 1 \Rightarrow \\
\qquad \mathit{BuildClusterEnvs}(c) = \langle \mathit{BuildClusterEnv}(\mathit{head}\ c) \rangle \frown \mathit{BuildClusterEnvs}(\mathit{tail}\ c)
\end{array}
$$

$$
\begin{array}{|l}
\mathit{BuildTierEnv} : \mathit{Tier} \rightarrow \mathit{TierEnv} \\
\hline
\forall \mathit{tier} : \operatorname{seq} \mathit{Cluster} \\
\quad \bullet \mathit{BuildTierEnv}(\mathit{tier}) = \langle\!| \mathit{Clusters} == \mathit{BuildClusterEnvs}(\mathit{tier}) |\!\rangle
\end{array}
$$

$$
\begin{array}{|l}
\mathit{BuildTierEnvs} : \operatorname{seq} \mathit{Tier} \rightarrow \operatorname{seq} \mathit{TierEnv} \\
\hline
\forall \mathit{tiers} : \operatorname{seq} \mathit{Tier} \bullet \\
\quad \mathit{BuildTierEnvs}(\mathit{tiers}) = \langle \mathit{BuildTierEnv}(\mathit{head}\ \mathit{tiers}) \rangle \frown \mathit{BuildTierEnvs}(\mathit{tail}\ \mathit{tiers})
\end{array}
$$

$$
\begin{array}{|l}
\mathit{BuildFWEnv} : \mathit{SCJProgram} \nrightarrow \mathit{FWEnv} \\
\hline
\mathit{dom}\ \mathit{BuildFWEnv} = \mathit{TranslatablePrograms} \\
\forall \mathit{scjProg} : \mathit{dom}\ \mathit{BuildFWEnv} \\
\quad \bullet \exists \mathit{tlmsID} : \mathit{Identifier};\ \mathit{tlmsBody} : \mathit{MissionSequencerClassBody}; \\
\qquad \mathit{tiers} : \operatorname{seq} \mathit{Tier} \mid \\
\qquad\quad \mathit{ProgTLMS}(\mathit{scjProg}) \neq \mathit{NoSequencer} \\
\qquad\qquad \Rightarrow \mathit{ProgTLMS}(\mathit{scjProg}) = \mathit{tlms}(\mathit{tlmsID}, \mathit{tlmsBody}) \\
\qquad\quad \bullet \mathit{BuildFWEnv}(\mathit{scjProg}) = \\
\qquad\qquad \langle\!| \mathit{TopLevelMS} == \mathit{tlmsID}, \mathit{Tiers} == \mathit{BuildTierEnvs}(\mathit{ProgTiers}(\mathit{scjProg})) |\!\rangle
\end{array}
$$

$$
\begin{array}{|l}
\hline
\mathit{BinderMethodEnv} \\
\mathit{MethodName} : N \\
\mathit{Locs} : \mathbb{F}\, N \\
\mathit{LocType} : \mathit{Type} \\
\mathit{Callers} : \mathbb{F}\, N \\
\mathit{CallerType} : \mathit{Type} \\
\mathit{ReturnType} : \mathit{Type} \\
\mathit{Params} : \operatorname{seq} \mathit{Type} \\
\mathit{Synchronised} : \mathbb{B} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\mathit{MCBEnv} == \mathbb{F}\, \mathit{BinderMethodEnv}
\end{array}
$$

$BuildBinderMethodName : N \rightarrow N$

$GetSFMethods : Safelet \rightarrow \operatorname{seq} ClassBodyDeclaration$

$\forall sf : Safelet$
$\quad \bullet GetSFMethods(sf) = sf.2.4$

$GetTLMSMethods : MissionSequencer \rightarrow \operatorname{seq} ClassBodyDeclaration$

$\forall tlms : MissionSequencer$
$\quad \bullet GetTLMSMethods(tlms) = tlms.2.2$

$SuperInterfacesOf : ClassDeclaration \rightarrow \mathbb{F}\, N$
$SuperClassOf : ClassDeclaration \rightarrow N$
$ProgramClasses : \mathbb{F}\, ClassDeclaration$
$ClassName : ClassDeclaration \rightarrow N$

$MethodsOf : ClassDeclaration \rightarrow \mathbb{F}\, MethodDeclaration$
$MethodName : MethodDeclaration \rightarrow N$
$TypeOf : ClassDeclaration \rightarrow Type$
$IsSync : MethodDeclaration \rightarrow \mathbb{B}$
$ReturnTypeOf : MethodDeclaration \rightarrow Type$
$MethodParams : MethodDeclaration \rightarrow \operatorname{seq} Type$

$ClassMethodMap : ClassDeclaration \rightarrow \mathbb{F}\, BinderMethodEnv$

$\forall c : ClassDeclaration$
$\quad \mid c \in ProgramClasses$
$\quad \bullet ClassMethodMap(c) =$
$\quad\quad \{m : MethodsOf(c)$
$\quad\quad\quad \bullet \langle\!| MethodName == MethodName(m),$
$\quad\quad\quad Locs == \{ClassName(c)\},$
$\quad\quad\quad LocType == TypeOf(c),$
$\quad\quad\quad Callers == \varnothing,$
$\quad\quad\quad CallerType == NullType,$
$\quad\quad\quad ReturnType == ReturnTypeOf(m),$
$\quad\quad\quad Params == MethodParams(m),$
$\quad\quad\quad Synchronised == IsSync(m) |\!\rangle\}$

$$GetCallerType : N \rightarrow Type$$

$$AddCaller : (BinderMethodEnv \times N) \rightarrow BinderMethodEnv$$

$$\begin{aligned}
&\forall\, meth : BinderMethodEnv;\ caller : N \\
&\quad \bullet\ AddCaller((meth, caller)) = \\
&\qquad \langle\!|\, MethodName == meth.MethodName, \\
&\qquad Locs == meth.Locs, \\
&\qquad LocType == meth.LocType, \\
&\qquad Callers == meth.Callers \cup \{caller\}, \\
&\qquad CallerType == GetCallerType(caller), \\
&\qquad ReturnType == meth.ReturnType, \\
&\qquad Params == meth.Params, \\
&\qquad Synchronised == meth.Synchronised \,|\!\rangle
\end{aligned}$$

$$IgnoredMethods : \mathbb{F}\, N$$

$$\begin{aligned}
&ClassesIn : SCJProgram \rightarrow \mathbb{F}\, ClassDeclaration \\
&IsMethodInvocation : BlockStatement \rightarrow \mathbb{B} \\
&CallTypeName : BlockStatement \nrightarrow N \\
&StatementsIn : MethodDeclaration \rightarrow \mathbb{F}\, BlockStatement \\
&NameOfMethod : MethodDeclaration \rightarrow N
\end{aligned}$$

$$MakesExternalMethodCall : ClassDeclaration \rightarrow \mathbb{B}$$

$$\begin{aligned}
&\forall\, c : ClassDeclaration \\
&\quad \bullet\ MakesExternalMethodCall(c) = \mathbf{True} \Leftrightarrow \\
&\qquad \exists\, m : MethodDeclaration \\
&\qquad\quad \bullet\ m \in MethodsOf(c) \\
&\qquad\quad \wedge\ NameOfMethod(m) \notin IgnoredMethods \\
&\qquad\quad \wedge\ \exists\, s : StatementsIn(m) \\
&\qquad\qquad \bullet\ IsMethodInvocation(s) = \mathbf{True}
\end{aligned}$$

$$LocOfExternalMethodCall : ClassDeclaration \rightarrow ClassDeclaration$$

$$\mathit{BuildMCBEnvs} : \mathit{SCJProgram} \pfun \mathit{MCBEnv}$$

$$\begin{array}{l}
\mathrm{dom}\ \mathit{BuildMCBEnvs} = \mathit{TranslatablePrograms} \\
\forall\, \mathit{scjProg} : \mathrm{dom}\ \mathit{BuildMCBEnvs} \\
\quad \bullet \forall\, c : \mathit{ClassesIn}(\mathit{scjProg}) \\
\qquad \bullet \exists\, \mathit{calledClass} : \mathit{ClassDeclaration} \\
\qquad \mid \mathit{calledClass} = \mathit{LocOfExternalMethodCall}(c) \\
\qquad \bullet \mathit{BuildMCBEnvs}(\mathit{scjProg}) = \\
\qquad\quad \{\mathit{bme} : \mathit{BinderMethodEnv} \\
\qquad\qquad \mid \mathit{bme} \in \mathit{ClassMethodMap}(\mathit{calledClass}) \\
\qquad\qquad\quad \bullet \mathit{AddCaller}(\mathit{bme}, \mathit{ClassName}(c))\}
\end{array}$$

$$\mathit{ThreadEnv} == (\mathit{ThreadIds} \times \mathit{Priority})$$

LockingEnv

$$\begin{array}{l}
\mathit{Threads} : \mathbb{F}\ \mathit{ThreadEnv} \\
\mathit{Objects} : \mathbb{F}\ \mathit{ObjectIds} \\
\mathit{Empty} : \mathbb{B}
\end{array}$$

$$\mathit{Empty} = \mathbf{True} \Leftrightarrow \mathit{Threads} = \varnothing \land \mathit{Objects} = \varnothing$$

$$\begin{array}{l}
\mathit{BuildThreads} : \mathbb{F}\ \mathit{ClassDeclaration} \to \mathbb{F}(\mathit{ThreadIds} \times \mathit{Priority}) \\
\mathit{BuildObjects} : \mathbb{F}\ \mathit{ClassDeclaration} \to \mathbb{F}\ \mathit{ObjectIds}
\end{array}$$

$$\mathit{BuildLockEnv} : \mathit{SCJProgram} \pfun \mathit{LockingEnv}$$

$$\begin{array}{l}
\mathrm{dom}\ \mathit{BuildLockEnv} = \mathit{TranslatablePrograms} \\
\forall\, \mathit{scjProg} : \mathit{SCJProgram} \\
\quad \bullet \exists\, \mathit{progClasses} : \mathbb{F}\ \mathit{ClassDeclaration};\ \mathit{threads} : \mathbb{F}\ \mathit{ThreadEnv}; \\
\qquad \mathit{objects} : \mathbb{F}\ \mathit{ObjectIds};\ \mathit{empty} : \mathbb{B} \\
\qquad \bullet \mathit{progClasses} = \mathit{ClassesIn}(\mathit{scjProg}) \\
\qquad \land\ \mathit{threads} = \mathit{BuildThreads}(\mathit{progClasses}) \\
\qquad \land\ \mathit{objects} = \mathit{BuildObjects}(\mathit{progClasses}) \\
\qquad \land\ (\mathit{threads} = \varnothing \land \mathit{objects} = \varnothing) \Rightarrow \mathit{empty} = \mathbf{True} \\
\qquad \land\ (\mathit{threads} \neq \varnothing \lor \mathit{objects} \neq \varnothing) \Rightarrow \mathit{empty} = \mathbf{False} \\
\qquad\quad \land\ \mathit{BuildLockEnv}(\mathit{scjProg}) = \\
\qquad \langle\!|\ \mathit{Threads} == \mathit{threads}, \qquad \mathit{Objects} == \mathit{objects}, \\
\qquad \mathit{Empty} == \mathit{empty}\ |\!\rangle
\end{array}$$

Generate Phase

This section presents the functions constituting the generate phase, which use the information in the environments to generate the *Circus* model for each component.

section $GeneratePhase$ **parents** $scj\_prelude, Framework, BuildPhase$

$$procNameOf : Process \rightarrow N$$

$$\begin{array}{l} ControlTierSync : CSExpression \\ MissionSync : CSExpression \\ SchedulablesSync : CSExpression \end{array}$$

$$\begin{array}{l} TierSync : TierEnv \times TierEnv \rightarrow CSExpression \\ \hline \forall t_1, t_2 : TierEnv \\ \quad \bullet \exists m : \operatorname{seq} N \\ \qquad \bullet TierSync(t) = cs(m) \end{array}$$

$$GetMissionID : ClusterEnv \rightarrow N$$

$$GenerateTiersFWProc : ClusterEnv \rightarrow Process$$

$$
\begin{array}{l}
GenerateClusterFWProcs : \mathrm{seq}\, ClusterEnv \rightarrow Process \\
\hline
\forall clusters : \mathrm{seq}\, ClusterEnv \\
\quad \bullet \# clusters = 1 \\
\qquad \Rightarrow GenerateClusterFWProcs(clusters) = \\
\qquad\quad procPar(\\
\qquad\qquad procName(GetMissionID(head\, clusters)), \\
\qquad\qquad MissionSync, \\
\qquad\qquad GenerateTiersFWProc(head\, clusters) \\
\qquad\quad) \\
\quad \wedge \# clusters \geq 1 \\
\qquad \Rightarrow GenerateClusterFWProcs(clusters) = \\
\qquad\quad procPar(\\
\qquad\qquad procPar(\\
\qquad\qquad\quad procName(GetMissionID(head\, clusters)), \\
\qquad\qquad\quad MissionSync, \\
\qquad\qquad\quad GenerateTiersFWProc(head\, clusters)), \\
\qquad\qquad SchedulablesSync, \\
\qquad\qquad GenerateClusterFWProcs(tail\, clusters) \\
\qquad\quad)
\end{array}
$$

$$
\begin{array}{l}
GenerateTierFWProcs : \mathrm{seq}\, TierEnv \rightarrow \mathrm{seq}\, Process \\
\hline
\forall tiers : \mathrm{seq}\, TierEnv \\
\quad \bullet \# tiers = 1 \Rightarrow \\
\qquad GenerateTierFWProcs(tiers) = \langle GenerateClusterFWProcs((head\, tiers).Clusters) \rangle \\
\quad \wedge \# tiers \geq 1 \Rightarrow \\
\qquad GenerateTierFWProcs(tiers) = \\
\qquad\quad \langle GenerateClusterFWProcs((head\, tiers).Clusters) \rangle \\
\qquad\quad {}^\frown GenerateTierFWProcs(tail\, tiers)
\end{array}
$$

$$GenerateTierFWProc : \mathrm{seq}\, TierEnv \rightarrow Process$$

$$
\begin{array}{l}
ControlTier : N \\
TopLevelMissionSequencerFWName : N
\end{array}
$$

$$GetParams : Identifier \rightarrow \mathrm{seq}\, Expression$$

$$
\begin{array}{|l}
GenerateFWProcs : FWEnv \rightarrow \mathrm{seq}\, Process \\
\hline
\forall\, env : FWEnv \\
\quad \bullet \exists\, fwProc : Process;\ controlTierProc : Process;\ tierProcs : \mathrm{seq}\, Process \\
\quad\quad |\ fwProc = procPar(\\
\quad\quad\quad procName(ControlTier), \\
\quad\quad\quad TierSync(head\ env.Tiers), \\
\quad\quad\quad GenerateTierFWProc(env.Tiers) \\
\quad\quad) \\
\quad\quad \land\ controlTierProc = procPar(\\
\quad\quad\quad procName(SafeletFWName), \\
\quad\quad\quad ControlTierSync, \\
\quad\quad\quad procInstP(procName(TopLevelMissionSequencerFWName), \\
\quad\quad\quad\quad GetParams(env.TopLevelMS)) \\
\quad\quad) \\
\quad\quad \land\ tierProcs = GenerateTierFWProcs(env.Tiers) \\
\quad\quad \bullet\ GenerateFWProcs(env) = \langle fwProc \rangle \frown \langle controlTierProc \rangle \frown tierProcs
\end{array}
$$

$$
\begin{array}{|l}
GenerateAppTierProcs : \mathrm{seq}\ TierAppEnv \rightarrow Process
\end{array}
$$

$$
\begin{array}{|l}
GenerateAppProc : AppProcEnv \rightarrow Process \\
\hline
\forall\, appProcEnv : AppProcEnv \\
\quad \bullet \exists\, sfAppEnv : AppEnv;\ tlmsAppEnv : AppEnv;\ tiersAppEnvs : \mathrm{seq}\ TierAppEnv \\
\quad\quad |\ sfAppEnv = GetSafeletAppEnv(appProcEnv) \\
\quad\quad \land\ tlmsAppEnv = GetTLMSAppEnv(appProcEnv) \\
\quad\quad \land\ tiersAppEnvs = GetTiersAppEnv(appProcEnv) \\
\quad\quad \bullet\ GenerateAppProc(appProcEnv) = \\
\quad\quad\quad procInter(\\
\quad\quad\quad\quad procInter(\\
\quad\quad\quad\quad\quad procInstP(procName(sfAppEnv.Name), sfAppEnv.Parameters), \\
\quad\quad\quad\quad\quad procInstP(procName(tlmsAppEnv.Name), tlmsAppEnv.Parameters) \\
\quad\quad\quad\quad), \\
\quad\quad\quad GenerateAppTierProcs(tiersAppEnvs) \\
\quad\quad)
\end{array}
$$

$$
\begin{array}{|l}
Locking : N \\
Threads : N \\
ThreadSync : CSExpression \\
Objects : N
\end{array}
$$

$BinderCallChan : N \rightarrow \operatorname{seq} N$

$NaturalCallChan : N \rightarrow \operatorname{seq} N$

$NaturalRetChan : N \rightarrow \operatorname{seq} N$

$BindeRetChan : N \rightarrow \operatorname{seq} N$

$MCBParams : \operatorname{seq} Type \rightarrow Expression$

$GenerateMCBChan : BinderMethodEnv \rightarrow CircusParagraph$

$$\begin{array}{l}
\forall\, bme : BinderMethodEnv \\
\quad \bullet\ GenerateMCBChan(bme) = chanDef(\\
\qquad multiDecl(chanNameWithType(NaturalCallChan(bme.MethodName), \\
\qquad\quad MCBParams(bme.Params)), \\
\qquad scDecl(chanNameWithType(NaturalRetChan(bme.MethodName), \\
\qquad\quad MCBParams(bme.Params)))) \\
\quad)
\end{array}$$

$MethodCallBinderSync : N$

$GenerateMethodCallBinderSync : \mathbb{P}\, BinderMethodEnv \rightarrow CircusParagraph$

$GenerateMCBChans : \mathbb{P}\, BinderMethodEnv \rightarrow \operatorname{seq} CircusParagraph$

$$\begin{array}{l}
\forall\, bEnvs : \mathbb{P}\, BinderMethodEnv \\
\quad \mid bEnvs \neq \varnothing \\
\quad \bullet\ \exists\, seqCP : \operatorname{seq} CircusParagraph \\
\qquad \mid \forall\, bme : bEnvs \bullet GenerateMCBChan(bme) \in ran\ seqCP \\
\qquad \bullet\ GenerateMCBChans(bEnvs) = seqCP
\end{array}$$

$BinderCallComm : N \rightarrow N$

$NaturalCallComm : N \rightarrow N$

$NaturalRetComm : N \rightarrow N$

$BindeRetComm : N \rightarrow N$

$GenerateMCBName : N \rightarrow N$

$BinderCallParams : \text{seq } Type \rightarrow \text{seq } CParameter$

$NaturalCallParams : \text{seq } Type \rightarrow \text{seq } CParameter$

$NaturalRetParams : \text{seq } Type \rightarrow \text{seq } CParameter$

$BinderRetParams : \text{seq } Type \rightarrow \text{seq } CParameter$

$BinderActions : N$
$DoneTLS : Communication$
$NoState : SchemaExp$
$MethodCallBinder : N$

$GenerateMCBAction : BinderMethodEnv \rightarrow \mathbb{P} Paragraph$

```
$\forall bme : BinderMethodEnv$
  $\bullet\ GenerateMCBAction(bme) = actDef(GenerateMCBName(bme.MethodName),$
    $prefixExp((BinderCallComm(bme.MethodName),$
        $BinderCallParams(bme.Params)),$
      $prefixExp((NaturalCallComm(bme.MethodName),$
          $BinderCallParams(bme.Params)),$
        $prefixExp((NaturalRetComm(bme.MethodName),$
            $BinderCallParams(bme.Params)),$
          $prefixExp((BindeRetComm(bme.MethodName),$
              $BinderCallParams(bme.Params)),$
            $actName(GenerateMCBName(bme.MethodName))$
          $)$
        $)$
      $)$
    $)$
  $)$
```

$$
\begin{array}{l}
GenerateMCBActions : \mathbb{P}\, BinderMethodEnv \rightarrow \mathrm{seq}\, PParagraph \\
\hline
\forall\, bEnvs : \mathbb{F}\, BinderMethodEnv \\
\quad \bullet \exists\, seqPP : \mathrm{seq}\, PParagraph \\
\qquad \mid \forall\, bme : bEnvs \bullet GenerateMCBAction(bme) \in ran\, seqPP \\
\qquad \bullet GenerateMCBActions(bEnvs) = seqPP
\end{array}
$$

$$
\begin{array}{l}
GenerateMCBProc : \mathbb{P}\, BinderMethodEnv \rightarrow CircusParagraph \\
\hline
\forall\, bmes : \mathbb{P}\, BinderMethodEnv \\
\quad \bullet GenerateMCBProc(bmes) = \\
\qquad procDef(pd(MethodCallBinder, \\
\qquad\quad proc(\\
\qquad\qquad \langle\rangle, \\
\qquad\qquad NoState, \\
\qquad\qquad GenerateMCBActions(bmes), \\
\qquad\qquad actInterupt(actName(BinderActions), prefixExp(DoneTLS, skip)) \\
\qquad\quad) \\
\qquad))
\end{array}
$$

$$
\begin{array}{l}
GenerateMCBModel : MCBEnv \rightarrow \mathrm{seq}\, CircusParagraph \\
\hline
\forall\, bEnvs : MCBEnv \\
\quad \bullet bEnvs = \varnothing \Rightarrow GenerateMCBModel(bEnvs) = \langle\rangle \\
\quad \wedge\, bEnvs \neq \varnothing \Rightarrow \\
\qquad GenerateMCBModel(bEnvs) = GenerateMCBChans(bEnvs) ^\frown \\
\qquad \langle GenerateMethodCallBinderSync(bEnvs), GenerateMCBProc(bEnvs) \rangle
\end{array}
$$

$$
\mid\; GenerateThreadProc : \mathbb{P}(ThreadIds \times Priority) \rightarrow Process
$$

$$
\mid\; GenerateObjectProc : \mathbb{P}\, ObjectIds \rightarrow Process
$$

$GenerateLockModel : LockingEnv \rightarrow \mathrm{seq}\ CircusParagraph$

```
∀ $lEnv : LockingEnv$
  • $lEnv.Empty = \mathbf{True}$
      $\Rightarrow GenerateLockModel(lEnv) = \langle\rangle$
  $\land$ $lEnv.Empty = \mathbf{False}$
      $\Rightarrow GenerateLockModel(lEnv) =$
      $\langle$
        $procDef(pd(Locking, procPar(procName(Threads),$
            $ThreadSync,$
            $procName(Objects)))$
        $),$
        $procDef(pd(Threads, GenerateThreadProc(lEnv.Threads))),$
        $procDef(pd(Objects, GenerateObjectProc(lEnv.Objects)))$
      $\rangle$
```

Translate SCJ Program

This section presents the *TransSCJProg* function, which translates an SCJ program into *Circus* using the functions defined in the previous sections.

section *TransSCJProg* **parents** *scj_prelude*, *SCJBNFEncoding*, *CircusBNFEncoding*, *BuildPhase*, *GeneratePhase*, *Framework*

$ProcessID : N \rightarrow ID$

$TransClasses : SCJProgram \rightarrow CircusProgram$

$FWName : N$
$AppName : N$
$MCBName : N$
$LockName : N$
$ProgName : Identifier \rightarrowtail\!\!\!\!\!\!+ N$

$appComms : CSExpression$
$mcbComms : CSExpression$
$lockComms : CSExpression$

$$
\begin{array}{l}
\mathit{TransSCJProg} : \mathit{Identifier} \times \mathit{SCJProgram} \pfun \mathit{CircusProgram} \\
\hline
\mathit{dom}\ \mathit{TransSCJProg} = \mathit{Identifier} \times \mathit{TranslatablePrograms} \\
\forall \mathit{name} : \mathit{Identifier};\ \mathit{scjProg} : \mathit{SCJProgram} \\
\quad \bullet \exists \mathit{app} : \mathit{CircusProgram};\ \mathit{program} : \mathit{CircusProgram}; \\
\qquad \mathit{fwProcs} : \mathrm{seq}\, \mathit{Process};\ \mathit{appProc} : \mathit{Process};\ \mathit{lockModel} : \mathrm{seq}\, \mathit{CircusParagraph}; \\
\qquad \mathit{mcbModel} : \mathrm{seq}\, \mathit{CircusParagraph};\ \mathit{fwEnv} : \mathit{FWEnv}; \\
\qquad \mathit{appEnv} : \mathit{AppProcEnv};\ \mathit{mcbEnvs} : \mathit{MCBEnv};\ \mathit{lockEnv} : \mathit{LockingEnv} \mid \\
\qquad\quad \mathit{fwEnv} = \mathit{BuildFWEnv}(\mathit{scjProg}) \\
\qquad\quad \mathit{appEnv} = \mathit{BuildAppProcEnv}(\mathit{scjProg}) \\
\qquad\quad \mathit{mcbEnvs} = \mathit{BuildMCBEnvs}(\mathit{scjProg}) \\
\qquad\quad \mathit{lockEnv} = \mathit{BuildLockEnv}(\mathit{scjProg}) \\
\qquad\quad \mathit{app} = \mathit{TransClasses}(\mathit{scjProg}) \\
\qquad\quad \land\ \mathit{fwProcs} = \mathit{GenerateFWProcs}(\mathit{fwEnv}) \\
\qquad\quad \land\ \mathit{appProc} = \mathit{GenerateAppProc}(\mathit{appEnv}) \\
\qquad\quad \land\ \mathit{mcbModel} = \mathit{GenerateMCBModel}(\mathit{mcbEnvs}) \\
\qquad\quad \land\ \mathit{lockModel} = \mathit{GenerateLockModel}(\mathit{lockEnv}) \\
\qquad\quad \land\ \mathit{program} = \langle \mathit{procDef}(\mathit{pd}(\mathit{ProgName}(\mathit{name}), \\
\qquad\quad \mathit{procHide}(\mathit{procPar}(\\
\qquad\qquad \mathit{procHide}(\\
\qquad\qquad\quad \mathit{procPar}(\\
\qquad\qquad\qquad \mathit{procName}(\mathit{FWName}), \\
\qquad\qquad\qquad \mathit{appComms}, \\
\qquad\qquad\qquad \mathit{procHide}(\\
\qquad\qquad\qquad\quad \mathit{procPar}(\mathit{procName}(\mathit{AppName}), \\
\qquad\qquad\qquad\quad \mathit{mcbComms}, \\
\qquad\qquad\qquad\quad \mathit{procName}(\mathit{MCBName})), \\
\qquad\qquad\qquad \mathit{mcbComms})), \\
\qquad\qquad\quad \mathit{appComms}), \\
\qquad\qquad \mathit{lockComms}, \\
\qquad\qquad \mathit{procName}(\mathit{LockName})), \\
\qquad\quad \mathit{lockComms})))\rangle\ \bullet \\
\qquad\quad \mathit{TransSCJProg}(\mathit{name}, \mathit{scjProg}) = \\
\qquad\qquad \mathit{framework} \cat \langle \mathit{procDef}(\mathit{pd}(\mathit{FWName}, \mathrm{head}\, \mathit{fwProcs}))\rangle \\
\qquad\qquad\quad \cat \mathit{app} \cat \langle \mathit{procDef}(\mathit{pd}(\mathit{AppName}, \mathit{appProc}))\rangle \\
\qquad\qquad\quad \cat \mathit{mcbModel} \cat \mathit{lockModel} \cat \mathit{program}
\end{array}
$$

Appendix F

Translated Application 1: Shared Buffer

This appendix contains the full model of the aircraft application (described in Sect. 2.1.2). Section F.1 contains the definitions of the process identifiers. Section F.2 contains the definition of top-level network processes and channels. Then we present the models of the paradigm objects: Sect. F.3 contains the safelet, Sect. F.4 contains the top-level mission sequencer, Sect. F.5 contains the mission, and Sect. F.5.1 contains its schedulables.

F.1 ID Definitions

F.1.1 MissionIds

section *MissionIds* **parents** *scj_prelude*, *MissionId*

$$
\begin{array}{|l}
MainMissionMID : MissionID \\
\hline
distinct\langle nullMissionId, MainMissionMID\rangle
\end{array}
$$

F.1.2 SchedulablesIds

section *SchedulableIds* **parents** *scj_prelude*, *SchedulableId*

$$
\begin{array}{|l}
MainMissionSequencerSID : SchedulableID \\
ProducerSID : SchedulableID \\
ConsumerSID : SchedulableID \\
\hline
distinct\langle nullSequencerId, nullSchedulableId, MainMissionSequencerSID, \\
ProducerSID, ConsumerSID\rangle
\end{array}
$$

F.1.3 NonParadignIds

section $\mathit{NonParadigmIds}$ **parents** $\mathit{scj\_prelude}, \mathit{GlobalTypes}$

$$
\begin{array}{|l}
\mathit{BufferID} : \mathit{NonParadigmID} \\
\hline
\mathit{distinct} \langle \mathit{BufferID} \rangle
\end{array}
$$

F.2 Network

F.2.1 Network Channel Sets

section $\mathit{NetworkChan}$ **parents** $\mathit{scj\_prelude}, \mathit{MissionId}, \mathit{MissionIds},$
$\mathit{SchedulableId}, \mathit{SchedulableIds}, \mathit{MissionChan}, \mathit{TopLevelMissionSequencerFWChan},$
$\mathit{FrameworkChan}, \mathit{SafeletChan}, \mathit{AperiodicEventHandlerChan}, \mathit{ManagedThreadChan},$
$\mathit{OneShotEventHandlerChan}, \mathit{PeriodicEventHandlerChan}, \mathit{MissionSequencerMethChan}$

channelset $\mathit{TerminateSync} ==$
$\{\!| \mathit{schedulables\_terminated}, \mathit{schedulables\_stopped}, \mathit{get\_activeSchedulables} |\!\}$

channelset $\mathit{ControlTierSync} ==$
$\{\!| \mathit{start\_toplevel\_sequencer}, \mathit{done\_toplevel\_sequencer}, \mathit{done\_safeletFW} |\!\}$

channelset $\mathit{TierSync} ==$
$\{\!| \mathit{start\_mission} . \mathit{PCMission}, \mathit{done\_mission} . \mathit{PCMission},$
$\mathit{done\_safeletFW}, \mathit{done\_toplevel\_sequencer} |\!\}$

channelset $\mathit{MissionSync} ==$
$\{\!| \mathit{done\_safeletFW}, \mathit{done\_toplevel\_sequencer}, \mathit{register},$
$\mathit{signalTerminationCall}, \mathit{signalTerminationRet}, \mathit{activate\_schedulables}, \mathit{done\_schedulable},$
$\mathit{cleanupSchedulableCall}, \mathit{cleanupSchedulableRet} |\!\}$

channelset $\mathit{SchedulablesSync} ==$
$\{\!| \mathit{activate\_schedulables}, \mathit{done\_safeletFW}, \mathit{done\_toplevel\_sequencer} |\!\}$

channelset $\mathit{ClusterSync} ==$
$\{\!| \mathit{done\_toplevel\_sequencer}, \mathit{done\_safeletFW} |\!\}$

channelset $\mathit{SafeltAppSync} \mathrel{\widehat=} \{\!| \mathit{getSequencerCall}, \mathit{getSequencerRet},$
$\mathit{initializeApplicationCall}, \mathit{initializeApplicationRet}, \mathit{end\_safelet\_app} |\!\}$

channelset $\mathit{MissionSequencerAppSync} ==$
$\{\!| \mathit{getNextMissionCall}, \mathit{getNextMissionRet}, \mathit{end\_sequencer\_app} |\!\}$

channelset $MissionAppSync ==$
$\lbrace\!| initializeCall, register, initializeRet, cleanupMissionCall, cleanupMissionRet |\!\rbrace$

channelset $AppSync ==$
$\bigcup\{SafeltAppSync, MissionSequencerAppSync, MissionAppSync,$
$MTAppSync, OSEHSync, APEHSync, PEHSync,$
$\lbrace\!| getSequencer, end\_mission\_app, end\_managedThread\_app,$
$setCeilingPriority, requestTerminationCall,$
$requestTerminationRet, terminationPendingCall,$
$terminationPendingRet, handleAsyncEventCall, handleAsyncEventRet |\!\rbrace\}$

channelset $ThreadSync ==$
$\lbrace\!| raise\_thread\_priority, lower\_thread\_priority,$
$isInterruptedCall, isInterruptedRet, get\_priorityLevel |\!\rbrace$

channelset $LockingSync ==$
$\lbrace\!| lockAcquired, startSyncMeth, endSyncMeth, waitCall, waitRet,$
$notify, isInterruptedCall, isInterruptedRet, interruptedCall, interruptedRet,$
$done\_toplevel\_sequencer, get\_priorityLevel |\!\rbrace$

F.2.2 MethodCallBinder

section $MethodCallBindingChannels$ **parents** $scj\_prelude, GlobalTypes,$
$FrameworkChan, MissionId, MissionIds, SchedulableId, SchedulableIds,$
$ThreadIds, NonParadigmIds$

channel $binder\_readCall : NonParadigmID \times SchedulableID \times ThreadID$
channel $binder\_readRet : NonParadigmID \times SchedulableID \times ThreadID \times \mathbb{Z}$

$readLocs == \{BufferID\}$
$readCallers == \{ConsumerSID\}$

channel $binder\_writeCall : NonParadigmID \times SchedulableID \times ThreadID \times \mathbb{Z}$
channel $binder\_writeRet : NonParadigmID \times SchedulableID \times ThreadID$

$writeLocs == \{BufferID\}$
$writeCallers == \{ProducerSID\}$

channelset $MethodCallBinderSync == \lbrace\!| done\_toplevel\_sequencer, binder\_readCall,$
$binder\_readRet, binder\_writeCall, binder\_writeRet |\!\rbrace$

section *NetworkMethodCallBinder* **parents** *scj_prelude*, *MissionId*, *MissionIds*, *SchedulableId*, *SchedulableIds*, *MethodCallBindingChannels*, *BuffeMethChan*, *PCMissionMethChan*

process *MethodCallBinder* $\widehat{=}$ **begin**

$$read\_MethodBinder \widehat{=}$$
$$\left(\begin{array}{l} binder\_readCall\,?\,loc : (loc \in readLocs) \\ \qquad ?\,caller : (caller \in readCallers)\,?\,callingThread \longrightarrow \\ readCall\,.\,loc\,.\,caller\,.\,callingThread \longrightarrow \\ readRet\,.\,loc\,.\,caller\,.\,callingThread\,?\,ret \longrightarrow \\ binder\_readRet\,.\,loc\,.\,caller\,.\,callingThread\,!\,ret \longrightarrow \\ read\_MethodBinder \end{array}\right)$$

$$write\_MethodBinder \widehat{=}$$
$$\left(\begin{array}{l} binder\_writeCall\,?\,loc : (loc \in writeLocs) \\ \qquad ?\,caller : (caller \in writeCallers)\,?\,callingThread\,?\,p1 \longrightarrow \\ writeCall\,.\,loc\,.\,caller\,.\,callingThread\,!\,p1 \longrightarrow \\ writeRet\,.\,loc\,.\,caller\,.\,callingThread \longrightarrow \\ binder\_writeRet\,.\,loc\,.\,caller\,.\,callingThread \longrightarrow \\ write\_MethodBinder \end{array}\right)$$

$$BinderActions \widehat{=}$$
$$\left(\begin{array}{l} read\_MethodBinder \\ ||| \\ write\_MethodBinder \end{array}\right)$$

$\bullet$ $BinderActions \vartriangle (done\_toplevel\_sequencer \longrightarrow \mathbf{Skip})$

end

F.2.3 Locking

section *NetworkLocking* **parents** *scj_prelude*, *GlobalTypes*, *FrameworkChan*, *MissionId*, *MissionIds*, *ThreadIds*, *NetworkChannels*, *ObjectFW*, *ThreadFW*, *Priority*

process *Threads* $\widehat{=}$
$$\Big(ThreadFW(ProducerTID, 10) \mathbin{|||} ThreadFW(ConsumerTID, 10)\Big)$$

process *Objects* $\widehat{=}$
$$\Big(ObjectFW(BufferOID)\Big)$$

process *Locking* $\widehat{=}$ $(Threads \llbracket ThreadSync \rrbracket Objects) \vartriangle (done\_toplevel\_sequencer \longrightarrow \mathbf{Skip})$

F.2.4 Program

$\mathbf{section}\ \mathit{NetworkProgram}\ \mathbf{parents}\ \mathit{scj\_prelude}, \mathit{MissionId}, \mathit{MissionIds}, \mathit{SchedulableId}, \mathit{SchedulableIds}, \mathit{MissionChan}, \mathit{SchedulableMethChan}, \mathit{MissionFW}, \mathit{SafeletFW}, \mathit{TopLevelMissionSequencerFW}, \mathit{NetworkChannels}, \mathit{ManagedThreadFW}, \mathit{SchedulableMissionSequencerFW}, \mathit{PeriodicEventHandlerFW}, \mathit{ObjectFW}, \mathit{AperiodicEventHandlerFW}, \mathit{OneShotEventHandlerFW}, \mathit{ThreadFW}, \mathit{PCSafeletApp}, \mathit{PCMissionSequencerApp}, \mathit{PCMissionApp}, \mathit{ProducerApp}, \mathit{ConsumerApp}$

$$\mathbf{process}\ \mathit{ControlTier} \mathrel{\widehat=} \left(\begin{array}{l} \mathit{SafeletFW} \\ \quad [\![\mathit{ControlTierSync}]\!] \\ \mathit{TopLevelMissionSequencerFW}(\mathit{PCMissionSequencer}) \end{array}\right)$$

$$\mathbf{process}\ \mathit{Tier0} \mathrel{\widehat=} \left(\begin{array}{l} \mathit{MissionFW}(\mathit{PCMissionID}) \\ \quad [\![\mathit{MissionSync}]\!] \\ \left(\begin{array}{l} \mathit{ManagedThreadFW}(\mathit{ProducerID}) \\ \quad [\![\mathit{SchedulablesSync}]\!] \\ \mathit{ManagedThreadFW}(\mathit{ConsumerID}) \end{array}\right) \end{array}\right)$$

$$\mathbf{process}\ \mathit{Framework} \mathrel{\widehat=} \left(\begin{array}{l} \mathit{ControlTier} \\ \quad [\![\mathit{TierSync}]\!] \\ \mathit{Tier0} \end{array}\right)$$

$$\mathbf{process}\ \mathit{Application} \mathrel{\widehat=} \left(\begin{array}{l} \mathit{PCSafeletApp} \\ ||| \\ \mathit{PCMissionSequencerApp} \\ ||| \\ \mathit{PCMissionApp} \\ ||| \\ \mathit{ProducerApp}(\mathit{PCMissionID}) \\ ||| \\ \mathit{ConsumerApp}(\mathit{PCMissionID}) \\ ||| \\ \mathit{BufferApp} \end{array}\right)$$

section *Network* **parents** *NetworkProgram*, *MethodCallBindingChannels*, *NetworkMethodCallBinder*, *NetworkChan*, *NetworkLocking*

process $Bound\_Application \mathrel{\widehat=} Application \llbracket\ MethodCallBinderSync\ \rrbracket\ MethodCallBinder$
process $Program \mathrel{\widehat=} \left(Framework\ \llbracket\ AppSync\ \rrbracket\ Bound\_Application\right)\ \llbracket\ LockingSync\ \rrbracket\ Locking$

F.3 BSafelet

section *BSafeletApp* **parents** *scj_prelude*, *SchedulableId*, *SchedulableIds*, *SafeletChan*, *MethodCallBindingChannels*

process $BSafeletApp \mathrel{\widehat=}$ **begin**

$$InitializeApplication \mathrel{\widehat=} \left(\begin{array}{l} initializeApplicationCall \longrightarrow \\ initializeApplicationRet \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$GetSequencer \mathrel{\widehat=} \left(\begin{array}{l} getSequencerCall \longrightarrow \\ getSequencerRet\,!\,MainMissionSequencerSID \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$Methods \mathrel{\widehat=} \left(\begin{array}{l} GetSequencer \\ \Box \\ InitializeApplication \end{array}\right);\ Methods$$

$\bullet\ (Methods) \mathbin{\triangle} (end\_safelet\_app \longrightarrow \mathbf{Skip})$

end

F.4 MainMissionSequencer

section *MainMissionSequencerApp* **parents** *TopLevelMissionSequencerChan*, *MissionId*, *MissionIds*, *SchedulableId*, *SchedulableIds*, *PCMissionSequencerClass*, *MethodCallBindingChannels*

process *MainMissionSequencerApp* $\widehat{=}$ **begin**

State

$this : \mathbf{ref}\ PCMissionSequencerClass$

state *State*

Init

$State'$

$this' = circnewPCMissionSequencerClass()$

$$GetNextMission \mathrel{\widehat{=}} \mathbf{var}\ ret : MissionID \bullet \left(\begin{array}{l} getNextMissionCall\,.\,PCMissionSequencerSID \longrightarrow \\ ret := this\,.\,getNextMission(); \\ getNextMissionRet\,.\,PCMissionSequencerSID\,!\,ret \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$Methods \mathrel{\widehat{=}} \left(GetNextMission\right) ;\ Methods$$

$$\bullet\ (Init\ ;\ Methods) \bigtriangleup (end\_sequencer\_app\,.\,PCMissionSequencerSID \longrightarrow \mathbf{Skip})$$

end

section *MainMissionSequencerClass* **parents** *scj_prelude*, *SchedulableId*, *SchedulableIds*, *SafeletChan*, *MethodCallBindingChannels*, *MissionId*, *MissionIds*

class *MainMissionSequencerClass* $\widehat{=}$ **begin**

state *State*

$returnedMission : \mathbb{B}$

state *State*

initial *Init*

$State'$

$returnedMission' = \mathbf{False}$

protected *getNextMission* $\widehat{=}$

$$\left(\begin{array}{l}\mathbf{if}\ (\neg\ returnedMission) \longrightarrow \\ \quad \left(\begin{array}{l} returnedMission := \mathbf{True}; \\ ret := PCMissionMID \end{array}\right) \\ [\!]\ \neg\ (\neg\ returnedMission) \longrightarrow \\ \quad \left(ret := nullMissionId\right) \\ \mathbf{fi}\end{array}\right)$$

• **Skip**

end

F.5 MainMission

section *MainMissionApp* **parents** *scj_prelude*, *MissionId*, *MissionIds*, *SchedulableId*, *SchedulableIds*, *MissionChan*, *SchedulableMethChan*, *MethodCallBindingChannels*

process *MainMissionApp* $\widehat{=}$ **begin**

$$InitializePhase \mathrel{\widehat{=}} \left(\begin{array}{l} initializeCall \,.\, PCMissionMID \longrightarrow \\ register \,!\, ProducerSID \,!\, PCMissionMID \longrightarrow \\ register \,!\, ConsumerSID \,!\, PCMissionMID \longrightarrow \\ initializeRet \,.\, PCMissionMID \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$CleanupPhase \mathrel{\widehat{=}} \mathbf{var}\ ret : \mathbb{B} \bullet \left(\begin{array}{l} cleanupMissionCall \,.\, PCMissionMID \longrightarrow \\ \left(ret := \mathbf{False}\right) ; \\ cleanupMissionRet \,.\, PCMissionMID \,!\, ret \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$Methods \mathrel{\widehat{=}} \left(\begin{array}{l} InitializePhase \\ \Box \\ CleanupPhase \end{array}\right) ;\ Methods$$

$\bullet\ (Methods) \vartriangle (end\_mission\_app \,.\, PCMissionMID \longrightarrow \mathbf{Skip})$

end

F.5.1 Schedulables of MainMission

$$\begin{array}{l}
\mathbf{section}\ ProducerApp\ \mathbf{parents}\ ManagedThreadChan, SchedulableId, SchedulableIds, \\
\quad MethodCallBindingChannels, MissionMethChan, BufferMethChan, ObjectIds, ThreadIds
\end{array}$$

$$\begin{array}{l}
\mathbf{process}\ ProducerApp \mathrel{\widehat=} \\
\quad pcMission : MissionID \bullet \mathbf{begin}
\end{array}$$

$$\begin{array}{l}
Run \mathrel{\widehat=} \\
\left(\begin{array}{l}
runCall \,.\, ProducerSID \longrightarrow \\
\left(\begin{array}{l}
\mathbf{var}\ i : \mathbb{Z} \bullet i := 1; \\
\mu X \bullet \\
\left(\begin{array}{l}
terminationPendingCall \,.\, pcMission \longrightarrow \\
terminationPendingRet \,.\, pcMission \,?\, terminationPending \longrightarrow \\
\mathbf{var}\ loopVar : \mathbb{B} \bullet loopVar := (\neg\ terminationPending); \\
\mathbf{if}\ (loopVar = \mathbf{True}) \longrightarrow \\
\qquad \left(\begin{array}{l}
binder\_writeCall \,.\, BufferID \,.\, ProducerSID \,.\, ProducerTID \,!\, i \longrightarrow \\
binder\_writeRet \,.\, BufferID \,.\, ProducerSID \,.\, ProducerTID \longrightarrow \\
\mathbf{Skip}; \\
i := i + 1; \\
\mathbf{if}\ (i \geq 5) \longrightarrow \\
\qquad \left(\begin{array}{l}
requestTerminationCall \,.\, pcMission \,.\, ProducerSID \longrightarrow \\
requestTerminationRet \,.\, pcMission \,.\, ProducerSID \,?\, rt \longrightarrow \\
\mathbf{Skip}
\end{array}\right) \\
[\!] \neg\ (i \geq 5) \longrightarrow \mathbf{Skip} \\
\mathbf{fi}
\end{array}\right) \\
[\!]\ (loopVar = \mathbf{False}) \longrightarrow \mathbf{Skip} \\
\mathbf{fi}
\end{array}\right) ;\ X
\end{array}\right) ; \\
runRet \,.\, ProducerSID \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}$$

$$\begin{array}{l}
Methods \mathrel{\widehat=} \\
\left(Run\right) ;\ Methods
\end{array}$$

$$\bullet\ (Methods) \mathrel{\triangle} (end\_managedThread\_app \,.\, ProducerSID \longrightarrow \mathbf{Skip})$$

end

section *ConsumerApp* **parents** *ManagedThreadChan*, *SchedulableId*, *SchedulableIds*, *MethodCallBindingChannels*, *MissionMethChan*, *BufferMethChan*, *ObjectIds*, *ThreadIds*

process *ConsumerApp* $\widehat{=}$
$pcMission : MissionID$ • **begin**

$$Run \widehat{=}$$
$$\left(\begin{array}{l}
runCall\,.\,ConsumerSID \longrightarrow \\
\left(\begin{array}{l}
\mu X \bullet \\
\left(\begin{array}{l}
terminationPendingCall\,.\,pcMission \longrightarrow \\
terminationPendingRet\,.\,pcMission\,?\,terminationPending \longrightarrow \\
\mathbf{var}\ loopVar : \mathbb{B} \bullet loopVar := (\neg\ terminationPending); \\
\mathbf{if}\ (loopVar = \mathbf{True}) \longrightarrow \\
\quad \left(\begin{array}{l}
\mathbf{var}\ result : \mathbb{Z} \bullet result := 999; \\
binder\_readCall\,.\,BufferID\,.\,ConsumerSID\,.\,ConsumerTID \longrightarrow \\
binder\_readRet\,.\,BufferID\,.\,ConsumerSID\,.\,ConsumerTID\,?\,b \longrightarrow \\
\mathbf{Skip}
\end{array}\right) \\
;\ X \\
[\!]\ (loopVar = \mathbf{False}) \longrightarrow \mathbf{Skip} \\
\mathbf{fi}
\end{array}\right)
\end{array}\right); \\
runRet\,.\,ConsumerSID \longrightarrow \\
\mathbf{Skip}
\end{array}\right)$$

$$Methods \widehat{=}$$
$$\left(Run\right)\ ;\ Methods$$

• $(Methods) \triangle (end\_managedThread\_app\,.\,ConsumerSID \longrightarrow \mathbf{Skip})$

end

Appendix G

Translated Application 2: Aircraft

This appendix contains the full model of the aircraft application (described in Sect. 2.1.3). Section G.1 contains the definitions of the process identifiers. Section G.2 contains the definition of top-level network processes and channels. Then we present the models of the paradigm objects: Sect. G.3 contains the safelet, Sect. G.4 contains the top-level mission sequencer, and Sect. G.5 contains each of the missions followed by its schedulables.

G.1 ID Files

G.1.1 MissionIds

$\mathbf{section}\ \mathit{MissionIds}\ \mathbf{parents}\ \mathit{scj\_prelude}, \mathit{MissionId}$

$$\begin{array}{|l}
\mathit{MainMissionMID} : \mathit{MissionID} \\
\mathit{TakeOffMissionMID} : \mathit{MissionID} \\
\mathit{CruiseMissionMID} : \mathit{MissionID} \\
\mathit{LandMissionMID} : \mathit{MissionID} \\
\hline
\mathit{distinct}\langle \mathit{nullMissionId}, \mathit{MainMissionMID}, \mathit{TakeOffMissionMID}, \\
\quad \mathit{CruiseMissionMID}, \mathit{LandMissionMID} \rangle
\end{array}$$

G.1.2 SchedulablesIds

section *SchedulableIds* **parents** *scj_prelude*, *SchedulableId*

$$
\begin{array}{|l}
MainMissionSequencerSID : SchedulableID \\
ACModeChangerSID : SchedulableID \\
EnvironmentMonitorSID : SchedulableID \\
ControlHandlerSID : SchedulableID \\
FlightSensorsMonitorSID : SchedulableID \\
CommunicationsHandlerSID : SchedulableID \\
AperiodicSimulatorSID : SchedulableID \\
LandingGearHandlerTakeOffSID : SchedulableID \\
TakeOffMonitorSID : SchedulableID \\
TakeOffFailureHandlerSID : SchedulableID \\
BeginLandingHandlerSID : SchedulableID \\
NavigationMonitorSID : SchedulableID \\
GroundDistanceMonitorSID : SchedulableID \\
LandingGearHandlerLandSID : SchedulableID \\
InstrumentLandingSystemMonitorSID : SchedulableID \\
SafeLandingHandlerSID : SchedulableID \\
\hline
distinct\langle nullSequencerId, nullSchedulableId, MainMissionSequencerSID, \\
ACModeChangerSID, EnvironmentMonitorSID, \\
ControlHandlerSID, FlightSensorsMonitorSID, \\
CommunicationsHandlerSID, AperiodicSimulatorSID, \\
LandingGearHandlerTakeOffSID, TakeOffMonitorSID, \\
TakeOffFailureHandlerSID, BeginLandingHandlerSID, \\
NavigationMonitorSID, GroundDistanceMonitorSID, \\
LandingGearHandlerLandSID, InstrumentLandingSystemMonitorSID, \\
SafeLandingHandlerSID\rangle
\end{array}
$$

G.1.3 ThreadIds

section *ThreadIds* **parents** *scj_prelude*, *GlobalTypes*

$$
\begin{array}{l}
InstrumentLandingSystemMonitorTID : ThreadID \\
SafeLandingHandlerTID : ThreadID \\
GroundDistanceMonitorTID : ThreadID \\
CommunicationsHandlerTID : ThreadID \\
ControlHandlerTID : ThreadID \\
AperiodicSimulatorTID : ThreadID \\
TakeOffFailureHandlerTID : ThreadID \\
LandingGearHandlerLandTID : ThreadID \\
EnvironmentMonitorTID : ThreadID \\
FlightSensorsMonitorTID : ThreadID \\
NavigationMonitorTID : ThreadID \\
ACModeChangerTID : ThreadID \\
BeginLandingHandlerTID : ThreadID \\
LandingGearHandlerTakeOffTID : ThreadID \\
TakeOffMonitorTID : ThreadID \\
\hline
distinct\langle SafeletTId, nullThreadId, \\
InstrumentLandingSystemMonitorTID, SafeLandingHandlerTID, \\
GroundDistanceMonitorTID, CommunicationsHandlerTID, \\
ControlHandlerTID, AperiodicSimulatorTID, \\
TakeOffFailureHandlerTID, LandingGearHandlerLandTID, \\
EnvironmentMonitorTID, FlightSensorsMonitorTID, \\
NavigationMonitorTID, ACModeChangerTID, \\
BeginLandingHandlerTID, LandingGearHandlerTakeOffTID, \\
TakeOffMonitorTID\rangle
\end{array}
$$

G.1.4 ObjectIds

section *ObjectIds* **parents** *scj_prelude*, *GlobalTypes*

$$
\begin{array}{l}
TakeOffMissionOID : ObjectID \\
LandMissionOID : ObjectID \\
\hline
distinct\langle TakeOffMissionOID, LandMissionOID\rangle
\end{array}
$$

G.2 Network

G.2.1 Network Channel Sets

section *NetworkChan* **parents** *scj_prelude*, *MissionId*, *MissionIds*,
SchedulableId, *SchedulableIds*, *MissionChan*, *TopLevelMissionSequencerFWChan*,
FrameworkChan, *SafeletChan*, *AperiodicEventHandlerChan*, *ManagedThreadChan*,
OneShotEventHandlerChan, *PeriodicEventHandlerChan*, *MissionSequencerMethChan*

channelset *TerminateSync* ==
$\{\!|$ *schedulables_terminated*, *schedulables_stopped*, *get_activeSchedulables* $|\!\}$

channelset *ControlTierSync* ==
$\{\!|$ *start_toplevel_sequencer*, *done_toplevel_sequencer*, *done_safeletFW* $|\!\}$

channelset *TierSync* ==
$\{\!|$ *start_mission . MainMission*, *done_mission . MainMission*,
done_safeletFW, *done_toplevel_sequencer* $|\!\}$

channelset *MissionSync* ==
$\{\!|$ *done_safeletFW*, *done_toplevel_sequencer*, *register*,
signalTerminationCall, *signalTerminationRet*, *activate_schedulables*, *done_schedulable*,
cleanupSchedulableCall, *cleanupSchedulableRet* $|\!\}$

channelset *SchedulablesSync* ==
$\{\!|$ *activate_schedulables*, *done_safeletFW*, *done_toplevel_sequencer* $|\!\}$

channelset *ClusterSync* ==
$\{\!|$ *done_toplevel_sequencer*, *done_safeletFW* $|\!\}$

channelset *SafeltAppSync* $\widehat{=}$ $\{\!|$ *getSequencerCall*, *getSequencerRet*,
initializeApplicationCall, *initializeApplicationRet*, *end_safelet_app* $|\!\}$

channelset *MissionSequencerAppSync* ==
$\{\!|$ *getNextMissionCall*, *getNextMissionRet*, *end_sequencer_app* $|\!\}$

channelset *MissionAppSync* ==
$\{\!|$ *initializeCall*, *register*, *initializeRet*, *cleanupMissionCall*, *cleanupMissionRet* $|\!\}$

channelset *AppSync* ==
$\bigcup\{$*SafeltAppSync*, *MissionSequencerAppSync*, *MissionAppSync*,
MTAppSync, *OSEHSync*, *APEHSync*, *PEHSync*,
$\{\!|$ *getSequencer*, *end_mission_app*, *end_managedThread_app*,
setCeilingPriority, *requestTerminationCall*,
requestTerminationRet, *terminationPendingCall*,
terminationPendingRet, *handleAsyncEventCall*, *handleAsyncEventRet* $|\!\}\}$

channelset $ThreadSync ==$
$\lbrace\!| raise\_thread\_priority, lower\_thread\_priority,$
$isInterruptedCall, isInterruptedRet, get\_priorityLevel |\!\rbrace$

channelset $LockingSync ==$
$\lbrace\!| lockAcquired, startSyncMeth, endSyncMeth, waitCall, waitRet,$
$notify, isInterruptedCall, isInterruptedRet, interruptedCall, interruptedRet,$
$done\_toplevel\_sequencer, get\_priorityLevel |\!\rbrace$

channelset $Tier0Sync == \lbrace\!| done\_toplevel\_sequencer, done\_safeletFW,$
$start\_mission . TakeOffMission, done\_mission . TakeOffMission,$
$initializeRet . TakeOffMission,$
$requestTermination . TakeOffMission . MainMissionSequencer,$
$start\_mission . CruiseMission, done\_mission . CruiseMission,$
$initializeRet . CruiseMission,$
$requestTermination . CruiseMission . MainMissionSequencer,$
$start\_mission . LandMission, done\_mission . LandMission,$
$initializeRet . LandMission,$
$requestTermination . LandMission . MainMissionSequencer |\!\rbrace$

G.2.2 MethodCallBinder

$\textbf{section}\ \mathit{MethodCallBindingChannels}\ \textbf{parents}\ \mathit{scj\_prelude}, \mathit{GlobalTypes}, \mathit{FrameworkChan},$
$\quad \mathit{MissionId}, \mathit{MissionIds}, \mathit{SchedulableId}, \mathit{SchedulableIds}, \mathit{ThreadIds}$

$\textbf{channel}\ \mathit{binder\_setCabinPressureCall} : \mathit{MissionID} \times \mathit{SchedulableID} \times \mathbb{R}$
$\textbf{channel}\ \mathit{binder\_setCabinPressureRet} : \mathit{MissionID} \times \mathit{SchedulableID}$

$\mathit{setCabinPressureLocs} == \{\mathit{MainMissionMID}\}$
$\mathit{setCabinPressureCallers} == \{\mathit{EnvironmentMonitorSID}\}$

$\textbf{channel}\ \mathit{binder\_setFuelRemainingCall} : \mathit{MissionID} \times \mathit{SchedulableID} \times \mathbb{R}$
$\textbf{channel}\ \mathit{binder\_setFuelRemainingRet} : \mathit{MissionID} \times \mathit{SchedulableID}$

$\mathit{setFuelRemainingLocs} == \{\mathit{MainMissionMID}\}$
$\mathit{setFuelRemainingCallers} == \{\mathit{EnvironmentMonitorSID}\}$

$\textbf{channel}\ \mathit{binder\_getAltitudeCall} : \mathit{MissionID} \times \mathit{SchedulableID}$
$\textbf{channel}\ \mathit{binder\_getAltitudeRet} : \mathit{MissionID} \times \mathit{SchedulableID} \times \mathbb{R}$

$\mathit{getAltitudeLocs} == \{\mathit{MainMissionMID}\}$
$\mathit{getAltitudeCallers} == \{\mathit{NavigationMonitorSID}, \mathit{TakeOffMonitorSID},$
$\quad \mathit{GroundDistanceMonitorSID}, \mathit{SafeLandingHandlerSID}\}$

$\textbf{channel}\ \mathit{binder\_setHeadingCall} : \mathit{MissionID} \times \mathit{SchedulableID} \times \mathbb{R}$
$\textbf{channel}\ \mathit{binder\_setHeadingRet} : \mathit{MissionID} \times \mathit{SchedulableID}$

$\mathit{setHeadingLocs} == \{\mathit{MainMissionMID}\}$
$\mathit{setHeadingCallers} == \{\mathit{FlightSensorsMonitorSID}\}$

$\textbf{channel}\ \mathit{binder\_stowLandingGearCall} : \mathit{MissionID} \times \mathit{SchedulableID}$
$\textbf{channel}\ \mathit{binder\_stowLandingGearRet} : \mathit{MissionID} \times \mathit{SchedulableID}$

$\mathit{stowLandingGearLocs} == \{\mathit{TakeOffMissionMID}, \mathit{LandMissionMID}\}$
$\mathit{stowLandingGearCallers} == \{\mathit{LandingGearHandlerSID}, \mathit{LandingGearHandlerLandSID}\}$

$\mathbf{channel}\ binder\_takeOffAbortCall : MissionID \times SchedulableID$
$\mathbf{channel}\ binder\_takeOffAbortRet : MissionID \times SchedulableID$

$takeOffAbortLocs == \{TakeOffMissionMID\}$
$takeOffAbortCallers == \{TakeOffFailureHandlerSID\}$

$\mathbf{channel}\ binder\_setAltitudeCall : MissionID \times SchedulableID \times \mathbb{R}$
$\mathbf{channel}\ binder\_setAltitudeRet : MissionID \times SchedulableID$

$setAltitudeLocs == \{MainMissionMID\}$
$setAltitudeCallers == \{FlightSensorsMonitorSID\}$

$\mathbf{channel}\ binder\_getHeadingCall : MissionID \times SchedulableID$
$\mathbf{channel}\ binder\_getHeadingRet : MissionID \times SchedulableID \times \mathbb{R}$

$getHeadingLocs == \{MainMissionMID\}$
$getHeadingCallers == \{NavigationMonitorSID\}$

$\mathbf{channel}\ binder\_getAirSpeedCall : MissionID \times SchedulableID$
$\mathbf{channel}\ binder\_getAirSpeedRet : MissionID \times SchedulableID \times \mathbb{R}$

$getAirSpeedLocs == \{MainMissionMID\}$
$getAirSpeedCallers == \{NavigationMonitorSID, TakeOffFailureHandlerSID\}$

$\mathbf{channel}\ binder\_deployLandingGearCall : MissionID \times SchedulableID$
$\mathbf{channel}\ binder\_deployLandingGearRet : MissionID \times SchedulableID$

$deployLandingGearLocs == \{TakeOffMissionMID, LandMissionMID\}$
$deployLandingGearCallers == \{LandingGearHandlerSID,$
$\quad LandingGearHandlerLandSID\}$

$\mathbf{channel}\ binder\_setEmergencyOxygenCall : MissionID \times SchedulableID \times \mathbb{R}$
$\mathbf{channel}\ binder\_setEmergencyOxygenRet : MissionID \times SchedulableID$

$setEmergencyOxygenLocs == \{MainMissionMID\}$
$setEmergencyOxygenCallers == \{EnvironmentMonitorSID\}$

channel $binder\_setAirSpeedCall : MissionID \times SchedulableID \times \mathbb{R}$
channel $binder\_setAirSpeedRet : MissionID \times SchedulableID$

$setAirSpeedLocs == \{MainMissionMID\}$
$setAirSpeedCallers == \{FlightSensorsMonitorSID\}$

channel $binder\_isLandingGearDeployedCall : MissionID \times SchedulableID$
channel $binder\_isLandingGearDeployedRet : MissionID \times SchedulableID \times \mathbb{B}$

$isLandingGearDeployedLocs == \{TakeOffMissionMID, LandMissionMID\}$
$isLandingGearDeployedCallers == \{LandingGearHandlerSID, LandingGearHandlerLandSID\}$

channelset $MethodCallBinderSync == \{\!| done\_toplevel\_sequencer,$
$binder\_setCabinPressureCall, binder\_setCabinPressureRet,$
$binder\_setFuelRemainingCall, binder\_setFuelRemainingRet,$
$binder\_getAltitudeCall, binder\_getAltitudeRet,$
$binder\_setHeadingCall, binder\_setHeadingRet,$
$binder\_stowLandingGearCall, binder\_stowLandingGearRet,$
$binder\_takeOffAbortCall, binder\_takeOffAbortRet,$
$binder\_setAltitudeCall, binder\_setAltitudeRet,$
$binder\_getHeadingCall, binder\_getHeadingRet,$
$binder\_getAirSpeedCall, binder\_getAirSpeedRet,$
$binder\_deployLandingGearCall, binder\_deployLandingGearRet,$
$binder\_setEmergencyOxygenCall, binder\_setEmergencyOxygenRet,$
$binder\_setAirSpeedCall, binder\_setAirSpeedRet,$
$binder\_isLandingGearDeployedCall, binder\_isLandingGearDeployedRet |\!\}$

section *NetworkMethodCallBinder* **parents** *scj_prelude*, *MissionId*, *MissionIds*, *SchedulableId*, *SchedulableIds*, *MethodCallBindingChannels*, *MainMissionMethChan*, *LandMissionMethChan*

process *MethodCallBinder* $\widehat{=}$ **begin**

$$setCabinPressure\_MethodBinder \mathrel{\widehat=} \left(\begin{array}{l} binder\_setCabinPressureCall \\ \quad ?\,loc : (loc \in setCabinPressureLocs) \\ \quad ?\,caller : (caller \in setCabinPressureCallers)\,?\,p1 \longrightarrow \\ setCabinPressureCall\,.\,loc\,.\,caller\,!\,p1 \longrightarrow \\ setCabinPressureRet\,.\,loc\,.\,caller \longrightarrow \\ binder\_setCabinPressureRet\,.\,loc\,.\,caller \longrightarrow \\ setCabinPressure\_MethodBinder \end{array}\right)$$

$$setFuelRemaining\_MethodBinder \mathrel{\widehat=} \left(\begin{array}{l} binder\_setFuelRemainingCall \\ \quad ?\,loc : (loc \in setFuelRemainingLocs) \\ \quad ?\,caller : (caller \in setFuelRemainingCallers)\,?\,p1 \longrightarrow \\ setFuelRemainingCall\,.\,loc\,.\,caller\,!\,p1 \longrightarrow \\ setFuelRemainingRet\,.\,loc\,.\,caller \longrightarrow \\ binder\_setFuelRemainingRet\,.\,loc\,.\,caller \longrightarrow \\ setFuelRemaining\_MethodBinder \end{array}\right)$$

$$getAltitude\_MethodBinder \mathrel{\widehat=} \left(\begin{array}{l} binder\_getAltitudeCall \\ \quad ?\,loc : (loc \in getAltitudeLocs) \\ \quad ?\,caller : (caller \in getAltitudeCallers) \longrightarrow \\ getAltitudeCall\,.\,loc\,.\,caller \longrightarrow \\ getAltitudeRet\,.\,loc\,.\,caller\,?\,ret \longrightarrow \\ binder\_getAltitudeRet\,.\,loc\,.\,caller\,!\,ret \longrightarrow \\ getAltitude\_MethodBinder \end{array}\right)$$

$$setHeading\_MethodBinder \mathrel{\widehat=} \left(\begin{array}{l} binder\_setHeadingCall \\ \quad ?\,loc : (loc \in setHeadingLocs) \\ \quad ?\,caller : (caller \in setHeadingCallers)\,?\,p1 \longrightarrow \\ setHeadingCall\,.\,loc\,.\,caller\,!\,p1 \longrightarrow \\ setHeadingRet\,.\,loc\,.\,caller \longrightarrow \\ binder\_setHeadingRet\,.\,loc\,.\,caller \longrightarrow \\ setHeading\_MethodBinder \end{array}\right)$$

$$stowLandingGear\_MethodBinder \mathrel{\widehat=} \\ \left(\begin{array}{l} binder\_stowLandingGearCall \\ \quad ?\, loc : (loc \in stowLandingGearLocs) \\ \quad ?\, caller : (caller \in stowLandingGearCallers) \longrightarrow \\ stowLandingGearCall \,.\, loc \,.\, caller \longrightarrow \\ stowLandingGearRet \,.\, loc \,.\, caller \longrightarrow \\ binder\_stowLandingGearRet \,.\, loc \,.\, caller \longrightarrow \\ stowLandingGear\_MethodBinder \end{array}\right)$$

$$takeOffAbort\_MethodBinder \mathrel{\widehat=} \\ \left(\begin{array}{l} binder\_takeOffAbortCall \\ \quad ?\, loc : (loc \in takeOffAbortLocs) \\ \quad ?\, caller : (caller \in takeOffAbortCallers) \longrightarrow \\ takeOffAbortCall \,.\, loc \,.\, caller \longrightarrow \\ takeOffAbortRet \,.\, loc \,.\, caller \longrightarrow \\ binder\_takeOffAbortRet \,.\, loc \,.\, caller \longrightarrow \\ takeOffAbort\_MethodBinder \end{array}\right)$$

$$setAltitude\_MethodBinder \mathrel{\widehat=} \\ \left(\begin{array}{l} binder\_setAltitudeCall \,?\, loc : (loc \in setAltitudeLocs) \\ \quad ?\, caller : (caller \in setAltitudeCallers) \,?\, p1 \longrightarrow \\ setAltitudeCall \,.\, loc \,.\, caller \,!\, p1 \longrightarrow \\ setAltitudeRet \,.\, loc \,.\, caller \longrightarrow \\ binder\_setAltitudeRet \,.\, loc \,.\, caller \longrightarrow \\ setAltitude\_MethodBinder \end{array}\right)$$

$$getHeading\_MethodBinder \mathrel{\widehat=} \\ \left(\begin{array}{l} binder\_getHeadingCall \,?\, loc : (loc \in getHeadingLocs) \\ \quad ?\, caller : (caller \in getHeadingCallers) \longrightarrow \\ getHeadingCall \,.\, loc \,.\, caller \longrightarrow \\ getHeadingRet \,.\, loc \,.\, caller \,?\, ret \longrightarrow \\ binder\_getHeadingRet \,.\, loc \,.\, caller \,!\, ret \longrightarrow \\ getHeading\_MethodBinder \end{array}\right)$$

$$getAirSpeed\_MethodBinder \mathrel{\widehat=} \\ \left(\begin{array}{l} binder\_getAirSpeedCall \\ \quad ?\, loc : (loc \in getAirSpeedLocs) \\ \quad ?\, caller : (caller \in getAirSpeedCallers) \longrightarrow \\ getAirSpeedCall \,.\, loc \,.\, caller \longrightarrow \\ getAirSpeedRet \,.\, loc \,.\, caller \,?\, ret \longrightarrow \\ binder\_getAirSpeedRet \,.\, loc \,.\, caller \,!\, ret \longrightarrow \\ getAirSpeed\_MethodBinder \end{array}\right)$$

$$deployLandingGear\_MethodBinder \mathrel{\widehat=}$$
$$\left(\begin{array}{l}
binder\_deployLandingGearCall \\
\quad ?\, loc : (loc \in deployLandingGearLocs) \\
\quad ?\, caller : (caller \in deployLandingGearCallers) \longrightarrow \\
deployLandingGearCall \,.\, loc \,.\, caller \longrightarrow \\
deployLandingGearRet \,.\, loc \,.\, caller \longrightarrow \\
binder\_deployLandingGearRet \,.\, loc \,.\, caller \longrightarrow \\
deployLandingGear\_MethodBinder
\end{array}\right)$$

$$setEmergencyOxygen\_MethodBinder \mathrel{\widehat=}$$
$$\left(\begin{array}{l}
binder\_setEmergencyOxygenCall \\
\quad ?\, loc : (loc \in setEmergencyOxygenLocs) \\
\quad ?\, caller : (caller \in setEmergencyOxygenCallers) \,?\, p1 \longrightarrow \\
setEmergencyOxygenCall \,.\, loc \,.\, caller \,!\, p1 \longrightarrow \\
setEmergencyOxygenRet \,.\, loc \,.\, caller \longrightarrow \\
binder\_setEmergencyOxygenRet \,.\, loc \,.\, caller \longrightarrow \\
setEmergencyOxygen\_MethodBinder
\end{array}\right)$$

$$setAirSpeed\_MethodBinder \mathrel{\widehat=}$$
$$\left(\begin{array}{l}
binder\_setAirSpeedCall \\
\quad ?\, loc : (loc \in setAirSpeedLocs) \\
\quad ?\, caller : (caller \in setAirSpeedCallers) \,?\, p1 \longrightarrow \\
setAirSpeedCall \,.\, loc \,.\, caller \,!\, p1 \longrightarrow \\
setAirSpeedRet \,.\, loc \,.\, caller \longrightarrow \\
binder\_setAirSpeedRet \,.\, loc \,.\, caller \longrightarrow \\
setAirSpeed\_MethodBinder
\end{array}\right)$$

$$isLandingGearDeployed\_MethodBinder \mathrel{\widehat=}$$
$$\left(\begin{array}{l}
binder\_isLandingGearDeployedCall \\
\quad ?\, loc : (loc \in isLandingGearDeployedLocs) \\
\quad ?\, caller : (caller \in isLandingGearDeployedCallers) \longrightarrow \\
isLandingGearDeployedCall \,.\, loc \,.\, caller \longrightarrow \\
isLandingGearDeployedRet \,.\, loc \,.\, caller \,?\, ret \longrightarrow \\
binder\_isLandingGearDeployedRet \,.\, loc \,.\, caller \,!\, ret \longrightarrow \\
isLandingGearDeployed\_MethodBinder
\end{array}\right)$$

$BinderActions \mathrel{\widehat=}$
$$\left(\begin{array}{l} setCabinPressure\_MethodBinder \\ ||| \\ setFuelRemaining\_MethodBinder \\ ||| \\ getAltitude\_MethodBinder \\ ||| \\ setHeading\_MethodBinder \\ ||| \\ stowLandingGear\_MethodBinder \\ ||| \\ takeOffAbort\_MethodBinder \\ ||| \\ setAltitude\_MethodBinder \\ ||| \\ getHeading\_MethodBinder \\ ||| \\ getAirSpeed\_MethodBinder \\ ||| \\ deployLandingGear\_MethodBinder \\ ||| \\ setEmergencyOxygen\_MethodBinder \\ ||| \\ setAirSpeed\_MethodBinder \\ ||| \\ isLandingGearDeployed\_MethodBinder \end{array}\right)$$

$\bullet\ BinderActions \triangle (done\_toplevel\_sequencer \longrightarrow \mathbf{Skip})$

end

G.2.3 Program

section *NetworkProgram* **parents** *scj_prelude*, *MissionId*, *MissionIds*, *SchedulableId*, *SchedulableIds*, *MissionChan*, *SchedulableMethChan*, *MissionFW*, *SafeletFW*, *TopLevelMissionSequencerFW*, *NetworkChannels*, *ManagedThreadFW*, *SchedulableMissionSequencerFW*, *PeriodicEventHandlerFW*, *OneShotEventHandlerFW*, *AperiodicEventHandlerFW*, *ObjectFW*, *ThreadFW*, *ACSafeletApp*, *MainMissionSequencerApp*, *MainMissionApp*, *ACModeChangerApp*, *ControlHandlerApp*, *CommunicationsHandlerApp*, *EnvironmentMonitorApp*, *FlightSensorsMonitorApp*, *TakeOffMissionApp*, *LandingGearHandlerApp*, *TakeOffFailureHandlerApp*, *TakeOffMonitorApp*, *CruiseMissionApp*, *BeginLandingHandlerApp*, *NavigationMonitorApp*, *LandMissionApp*, *LandingGearHandlerLandApp*, *SafeLandingHandlerApp*, *GroundDistanceMonitorApp*, *InstrumentLandingSystemMonitorApp*

$$\textbf{process}\ \mathit{ControlTier} \mathrel{\widehat=} \left(\begin{array}{l} \mathit{SafeletFW} \\ \quad [\![\mathit{ControlTierSync}]\!] \\ \mathit{TopLevelMissionSequencerFW}(\mathit{MainMissionSequencer}) \end{array}\right)$$

$$\textbf{process}\ \mathit{Tier0} \mathrel{\widehat=} \left(\begin{array}{l} \mathit{MissionFW}(\mathit{MainMissionID}) \\ \quad [\![\mathit{MissionSync}]\!] \\ \left(\begin{array}{l} \mathit{SchedulableMissionSequencerFW}(\mathit{ACModeChanger2ID}) \\ \quad [\![\mathit{SchedulablesSync}]\!] \\ \mathit{AperiodicEventHandlerFW}(\mathit{ControlHandlerID}, \mathit{aperiodic}, \\ \qquad (\mathit{time}(10, 0), \mathit{nullSchedulableId})) \\ \quad [\![\mathit{SchedulablesSync}]\!] \\ \mathit{AperiodicEventHandlerFW}(\mathit{CommunicationsHandlerID}, \mathit{aperiodic}, \\ \qquad (\mathit{NULL}, \mathit{nullSchedulableId})) \\ \quad [\![\mathit{SchedulablesSync}]\!] \\ \mathit{PeriodicEventHandlerFW}(\mathit{EnvironmentMonitorID}, \\ \qquad (\mathit{time}(10, 0), \mathit{NULL}, \mathit{NULL}, \mathit{nullSchedulableId})) \\ \quad [\![\mathit{SchedulablesSync}]\!] \\ \mathit{PeriodicEventHandlerFW}(\mathit{FlightSensorsMonitorID}, \\ \qquad (\mathit{time}(10, 0), \mathit{NULL}, \mathit{NULL}, \mathit{nullSchedulableId})) \end{array}\right) \end{array}\right)$$

process $Tier1 \mathrel{\widehat{=}}$

$$\left(\begin{array}{l}
\left(\begin{array}{l}
MissionFW(TakeOffMissionID) \\
\quad \llbracket MissionSync \rrbracket \\
\left(\begin{array}{l}
AperiodicEventHandlerFW(LandingGearHandlerID, aperiodic, \\
\qquad (NULL, nullSchedulableId)) \\
\quad \llbracket SchedulablesSync \rrbracket \\
AperiodicEventHandlerFW(TakeOffFailureHandlerID, aperiodic, \\
\qquad (NULL, nullSchedulableId)) \\
\quad \llbracket SchedulablesSync \rrbracket \\
PeriodicEventHandlerFW(TakeOffMonitorID, \\
\qquad (time(0,0), time(500,0), NULL, nullSchedulableId))
\end{array}\right)
\end{array}\right) \\
\quad \llbracket ClusterSync \rrbracket \\
\left(\begin{array}{l}
MissionFW(CruiseMissionID) \\
\quad \llbracket MissionSync \rrbracket \\
\left(\begin{array}{l}
AperiodicEventHandlerFW(BeginLandingHandlerID, aperiodic, \\
\qquad (NULL, nullSchedulableId)) \\
\quad \llbracket SchedulablesSync \rrbracket \\
PeriodicEventHandlerFW(NavigationMonitorID, \\
\qquad (time(0,0), time(10,0), NULL, nullSchedulableId))
\end{array}\right)
\end{array}\right) \\
\quad \llbracket ClusterSync \rrbracket \\
\left(\begin{array}{l}
MissionFW(LandMissionID) \\
\quad \llbracket MissionSync \rrbracket \\
\left(\begin{array}{l}
AperiodicEventHandlerFW(LandingGearHandlerLandID, aperiodic, \\
\qquad (NULL, nullSchedulableId)) \\
\quad \llbracket SchedulablesSync \rrbracket \\
AperiodicEventHandlerFW(SafeLandingHandlerID, aperiodic, \\
\qquad (NULL, nullSchedulableId)) \\
\quad \llbracket SchedulablesSync \rrbracket \\
PeriodicEventHandlerFW(GroundDistanceMonitorID, \\
\qquad (time(0,0), time(10,0), NULL, nullSchedulableId)) \\
\quad \llbracket SchedulablesSync \rrbracket \\
PeriodicEventHandlerFW(InstrumentLandingSystemMonitorID, \\
\qquad (time(0,0), time(10,0), NULL, nullSchedulableId))
\end{array}\right)
\end{array}\right)
\end{array}\right.$$

process $Framework \mathrel{\widehat{=}}$

$$\left(\begin{array}{l}
ControlTier \\
\quad \llbracket TierSync \rrbracket \\
\left(\begin{array}{l}
Tier0 \\
\quad \llbracket Tier0Sync \rrbracket \\
Tier1
\end{array}\right)
\end{array}\right)$$

$$\textbf{process}\ \mathit{Application} \mathrel{\widehat=}$$

$$\left(\begin{array}{l}
\mathit{ACSafeletApp} \\
\interleave \\
\mathit{MainMissionSequencerApp} \\
\interleave \\
\mathit{MainMissionApp} \\
\interleave \\
\mathit{ACModeChanger2App}(\mathit{MainMissionID}) \\
\interleave \\
\mathit{ControlHandlerApp} \\
\interleave \\
\mathit{CommunicationsHandlerApp} \\
\interleave \\
\mathit{EnvironmentMonitorApp}(\mathit{MainMissionID}) \\
\interleave \\
\mathit{FlightSensorsMonitorApp}(\mathit{MainMissionID}) \\
\interleave \\
\mathit{TakeOffMissionApp} \\
\interleave \\
\mathit{LandingGearHandlerApp}(\mathit{TakeOffMissionID}) \\
\interleave \\
\mathit{TakeOffFailureHandlerApp}(\mathit{TakeOffMissionID}, 10.0) \\
\interleave \\
\mathit{TakeOffMonitorApp}(\mathit{TakeOffMissionID}, 10.0, \mathit{landingGearHandlerID}) \\
\interleave \\
\mathit{CruiseMissionApp} \\
\interleave \\
\mathit{BeginLandingHandlerApp} \\
\interleave \\
\mathit{NavigationMonitorApp} \\
\interleave \\
\mathit{LandMissionApp} \\
\interleave \\
\mathit{LandingGearHandlerLandApp}(\mathit{LandMissionID}) \\
\interleave \\
\mathit{SafeLandingHandlerApp}(10.0) \\
\interleave \\
\mathit{GroundDistanceMonitorApp}(0.0) \\
\interleave \\
\mathit{InstrumentLandingSystemMonitorApp}(\mathit{LandMissionID})
\end{array}\right)$$

section *Network* **parents** *NetworkProgram*, *MethodCallBindingChannels*,
NetworkMethodCallBinder, *NetworkChan*, *NetworkLocking*

process $Bound\_Application \mathrel{\widehat=} Application \llbracket MethodCallBinderSync \rrbracket MethodCallBinder$
process $Program \mathrel{\widehat=} Framework \llbracket AppSync \rrbracket Bound\_Application$

G.3 ACSafelet

section *ACSafeletApp* **parents** *scj_prelude*, *SchedulableId*, *SchedulableIds*, *SafeletChan*,
MethodCallBindingChannels

process *ACSafeletApp* $\mathrel{\widehat=}$ **begin**

$$InitializeApplication \mathrel{\widehat=} \left(\begin{array}{l} initializeApplicationCall \longrightarrow \\ initializeApplicationRet \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$GetSequencer \mathrel{\widehat=} \left(\begin{array}{l} getSequencerCall \longrightarrow \\ getSequencerRet\,!\,MainMissionSequencerSID \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$Methods \mathrel{\widehat=} \left(\begin{array}{l} GetSequencer \\ \Box \\ InitializeApplication \end{array}\right) ;\ Methods$$

$\bullet\ (Methods) \bigtriangleup (end\_safelet\_app \longrightarrow \mathbf{Skip})$

end

G.4 MainMissionSequencer

section $\mathit{MainMissionSequencerApp}$ **parents** $\mathit{TopLevelMissionSequencerChan}$, $\mathit{MissionId}$, $\mathit{MissionIds}$, $\mathit{SchedulableId}$, $\mathit{SchedulableIds}$, $\mathit{MainMissionSequencerClass}$, $\mathit{MethodCallBindingChannels}$

process $\mathit{MainMissionSequencerApp} \mathrel{\widehat=}$ **begin**

$$\begin{array}{|l}
\textit{State} \\
\hline
\mathit{this} : \mathbf{ref}\ \mathit{MainMissionSequencerClass} \\
\hline
\end{array}$$

state State

$$\begin{array}{|l}
\textit{Init} \\
\hline
\mathit{State}' \\
\hline
\mathit{this}' = \mathbf{new}\ \mathit{MainMissionSequencerClass}() \\
\hline
\end{array}$$

$$\mathit{GetNextMission} \mathrel{\widehat=} \mathbf{var}\ \mathit{ret} : \mathit{MissionID} \bullet$$
$$\left(\begin{array}{l}
\mathit{getNextMissionCall} \,.\, \mathit{MainMissionSequencerSID} \longrightarrow \\
\left(\begin{array}{l}
\mathit{ret} := \mathit{this} \,.\, \mathit{getNextMission}(); \\
\mathit{getNextMissionRet} \,.\, \mathit{MainMissionSequencerSID}\,!\,\mathit{ret} \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}\right)$$

$$\mathit{Methods} \mathrel{\widehat=}$$
$$\Big(\mathit{GetNextMission}\Big) ;\ \mathit{Methods}$$

$$\bullet\ (\mathit{Init} ;\ \mathit{Methods}) \mathbin{\triangle} (\mathit{end\_sequencer\_app} \,.\, \mathit{MainMissionSequencerSID} \longrightarrow \mathbf{Skip})$$

end

section *MainMissionSequencerClass* **parents** *scj_prelude*, *SchedulableId*, *SchedulableIds*, *SafeletChan*, *MethodCallBindingChannels*, *MissionId*, *MissionIds*

class $MainMissionSequencerClass \mathrel{\widehat=}$ **begin**

state *State*

$returnedMission : \mathbb{B}$

state *State*

initial *Init*

$State~'$

$returnedMission' = \mathbf{False}$

protected $getNextMission \mathrel{\widehat=}$

$$\left(\begin{array}{l}\mathbf{if}\ (\neg\ returnedMission) \longrightarrow \\ \quad \left(\begin{array}{l} returnedMission := \mathbf{True}; \\ ret := MainMissionMID \end{array}\right) \\ [\!] \neg\ (\neg\ returnedMission) \longrightarrow \\ \quad \left(ret := nullMissionId\right) \\ \mathbf{fi}\end{array}\right)$$

$\bullet$ **Skip**

end

G.5 Missions

G.5.1 MainMission

section *MainMissionApp* **parents** *scj_prelude*, *GlobalTypes*, *MissionId*, *MissionIds*, *SchedulableId*, *SchedulableIds*, *MissionChan*, *SchedulableMethChan*, *MainMissionMethChan*, *MainMissionClass*, *MethodCallBindingChannels*

process *MainMissionApp* $\widehat{=}$ **begin**

State
$$this : \mathbf{ref}\ MainMissionClass$$

state *State*

Init
$$State'$$
$$this' = \mathbf{new}\ MainMissionClass()$$

$$InitializePhase \mathrel{\widehat{=}} \left(\begin{array}{l} initializeCall\,.\,MainMissionMID \longrightarrow \\ register\,!\,ACModeChanger2SID\,!\,MainMissionMID \longrightarrow \\ register\,!\,EnvironmentMonitorSID\,!\,MainMissionMID \longrightarrow \\ register\,!\,ControlHandlerSID\,!\,MainMissionMID \longrightarrow \\ register\,!\,FlightSensorsMonitorSID\,!\,MainMissionMID \longrightarrow \\ register\,!\,CommunicationsHandlerSID\,!\,MainMissionMID \longrightarrow \\ initializeRet\,.\,MainMissionMID \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$CleanupPhase \mathrel{\widehat{=}} \left(\begin{array}{l} cleanupMissionCall\,.\,MainMissionMID \longrightarrow \\ cleanupMissionRet\,.\,MainMissionMID\,!\,\mathbf{True} \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$getAirSpeedMeth \mathrel{\widehat{=}} \mathbf{var}\ ret : \mathbb{R} \bullet \left(\begin{array}{l} getAirSpeedCall\,.\,MainMissionMID\,?\,caller \longrightarrow \\ \left(\begin{array}{l} ret := this\,.\,getAirSpeed(); \\ getAirSpeedRet\,.\,MainMissionMID\,.\,caller\,!\,ret \longrightarrow \\ \mathbf{Skip} \end{array}\right) \end{array}\right)$$

$$
\begin{array}{l}
getAltitudeMeth \mathrel{\widehat=} \mathbf{var}\ ret : \mathbb{R} \bullet \\
\left(\begin{array}{l}
getAltitudeCall\,.\,MainMissionMID\,?\,caller \longrightarrow \\
\left(\begin{array}{l}
ret := this\,.\,getAltitude(); \\
getAltitudeRet\,.\,MainMissionMID\,.\,caller\,!\,ret \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
getCabinPressureMeth \mathrel{\widehat=} \mathbf{var}\ ret : \mathbb{R} \bullet \\
\left(\begin{array}{l}
getCabinPressureCall\,.\,MainMissionMID \longrightarrow \\
\left(\begin{array}{l}
ret := this\,.\,getCabinPressure(); \\
getCabinPressureRet\,.\,MainMissionMID\,!\,ret \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
getEmergencyOxygenMeth \mathrel{\widehat=} \mathbf{var}\ ret : \mathbb{R} \bullet \\
\left(\begin{array}{l}
getEmergencyOxygenCall\,.\,MainMissionMID \longrightarrow \\
\left(\begin{array}{l}
ret := this\,.\,getEmergencyOxygen(); \\
getEmergencyOxygenRet\,.\,MainMissionMID\,!\,ret \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
getFuelRemainingMeth \mathrel{\widehat=} \mathbf{var}\ ret : \mathbb{R} \bullet \\
\left(\begin{array}{l}
getFuelRemainingCall\,.\,MainMissionMID \longrightarrow \\
\left(\begin{array}{l}
ret := this\,.\,getFuelRemaining(); \\
getFuelRemainingRet\,.\,MainMissionMID\,!\,ret \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
getHeadingMeth \mathrel{\widehat=} \mathbf{var}\ ret : \mathbb{R} \bullet \\
\left(\begin{array}{l}
getHeadingCall\,.\,MainMissionMID\,?\,caller \longrightarrow \\
\left(\begin{array}{l}
ret := this\,.\,getHeading(); \\
getHeadingRet\,.\,MainMissionMID\,.\,caller\,!\,ret \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}\right)
\end{array}
$$

$$
setAirSpeedMeth \mathrel{\widehat=} \\
\left(\begin{array}{l}
setAirSpeedCall \,.\, MainMissionMID \,?\, caller \,?\, newAirSpeed \longrightarrow \\
this \,.\, setAirSpeed(newAirSpeed); \\
setAirSpeedRet \,.\, MainMissionMID \,.\, caller \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
$$

$$
setAltitudeMeth \mathrel{\widehat=} \\
\left(\begin{array}{l}
setAltitudeCall \,.\, MainMissionMID \,?\, caller \,?\, newAltitude \longrightarrow \\
this \,.\, setAltitude(newAltitude); \\
setAltitudeRet \,.\, MainMissionMID \,.\, caller \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
$$

$$
setCabinPressureMeth \mathrel{\widehat=} \\
\left(\begin{array}{l}
setCabinPressureCall \,.\, MainMissionMID \,?\, caller \,?\, newCabinPressure \longrightarrow \\
this \,.\, setCabinPressure(newCabinPressure); \\
setCabinPressureRet \,.\, MainMissionMID \,.\, caller \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
$$

$$
setEmergencyOxygenMeth \mathrel{\widehat=} \\
\left(\begin{array}{l}
setEmergencyOxygenCall \,.\, MainMissionMID \,?\, caller \\
\quad ?\, newEmergencyOxygen \longrightarrow \\
this \,.\, setEmergencyOxygen(newEmergencyOxygen); \\
setEmergencyOxygenRet \,.\, MainMissionMID \,.\, caller \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
$$

$$
setFuelRemainingMeth \mathrel{\widehat=} \\
\left(\begin{array}{l}
setFuelRemainingCall \,.\, MainMissionMID \,?\, caller \,?\, newFuelRemaining \longrightarrow \\
this \,.\, setFuelRemaining(newFuelRemaining); \\
setFuelRemainingRet \,.\, MainMissionMID \,.\, caller \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
$$

$$
setHeadingMeth \mathrel{\widehat=} \\
\left(\begin{array}{l}
setHeadingCall \,.\, MainMissionMID \,?\, caller \,?\, newHeading \longrightarrow \\
this \,.\, setHeading(newHeading); \\
setHeadingRet \,.\, MainMissionMID \,.\, caller \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
$$

$$Methods \mathrel{\widehat=} \left(\begin{array}{l} InitializePhase \\ \Box \\ CleanupPhase \\ \Box \\ getAirSpeedMeth \\ \Box \\ getAltitudeMeth \\ \Box \\ getCabinPressureMeth \\ \Box \\ getEmergencyOxygenMeth \\ \Box \\ getFuelRemainingMeth \\ \Box \\ getHeadingMeth \\ \Box \\ setAirSpeedMeth \\ \Box \\ setAltitudeMeth \\ \Box \\ setCabinPressureMeth \\ \Box \\ setEmergencyOxygenMeth \\ \Box \\ setFuelRemainingMeth \\ \Box \\ setHeadingMeth \end{array}\right) ; ~ Methods$$

$\bullet$ $(Init ~;~ Methods) \vartriangle (end\_mission\_app \,.\, MainMissionMID \longrightarrow \mathbf{Skip})$

end

section $MainMissionClass$ **parents** $scj\_prelude, SchedulableId, SchedulableIds,$
$SafeletChan, MethodCallBindingChannels$

class $MainMissionClass \widehat{=}$ **begin**

state $State$

$cabinPressure : \mathbb{R}$

$emergencyOxygen : \mathbb{R}$

$fuelRemaining : \mathbb{R}$

$altitude : \mathbb{R}$

$airSpeed : \mathbb{R}$

$heading : \mathbb{R}$

state $State$

initial $Init$

$State'$

public $getAirSpeed \widehat{=}$
$\left(ret := airSpeed\right)$

public $getAltitude \widehat{=}$
$\left(ret := altitude\right)$

public $getCabinPressure \widehat{=}$
$\left(ret := cabinPressure\right)$

public $getEmergencyOxygen \widehat{=}$
$\left(ret := emergencyOxygen\right)$

public $getFuelRemaining \widehat{=}$
$\left(ret := fuelRemaining\right)$

public $getHeading \widehat{=}$
$\left(ret := heading\right)$

public $setAirSpeed \mathrel{\widehat=} \mathbf{var}\ newAirSpeed : \mathbb{R} \bullet$
$\Big(airSpeed := newAirSpeed\Big)$

public $setAltitude \mathrel{\widehat=} \mathbf{var}\ newAltitude : \mathbb{R} \bullet$
$\Big(altitude := newAltitude\Big)$

public $setCabinPressure \mathrel{\widehat=} \mathbf{var}\ newCabinPressure : \mathbb{R} \bullet$
$\Big(cabinPressure := newCabinPressure\Big)$

public $setEmergencyOxygen \mathrel{\widehat=} \mathbf{var}\ newEmergencyOxygen : \mathbb{R} \bullet$
$\Big(emergencyOxygen := newEmergencyOxygen\Big)$

public $setFuelRemaining \mathrel{\widehat=} \mathbf{var}\ newFuelRemaining : \mathbb{R} \bullet$
$\Big(fuelRemaining := newFuelRemaining\Big)$

public $setHeading \mathrel{\widehat=} \mathbf{var}\ newHeading : \mathbb{R} \bullet$
$\Big(heading := newHeading\Big)$

$\bullet$ **Skip**

end

section *MainMissionMethChan* **parents** *GlobalTypes*, *MissionId*, *SchedulableId*

channel $getAirSpeedCall : MissionID \times SchedulableID$

channel $getAirSpeedRet : MissionID \times SchedulableID \times \mathbb{R}$

channel $getAltitudeCall : MissionID \times SchedulableID$

channel $getAltitudeRet : MissionID \times SchedulableID \times \mathbb{R}$

channel $getCabinPressureCall : MissionID$

channel $getCabinPressureRet : MissionID \times \mathbb{R}$

channel $getEmergencyOxygenCall : MissionID$

channel $getEmergencyOxygenRet : MissionID \times \mathbb{R}$

channel $getFuelRemainingCall : MissionID$

channel $getFuelRemainingRet : MissionID \times \mathbb{R}$

channel $getHeadingCall : MissionID \times SchedulableID$

channel $getHeadingRet : MissionID \times SchedulableID \times \mathbb{R}$

channel $setAirSpeedCall : MissionID \times SchedulableID \times \mathbb{R}$

channel $setAirSpeedRet : MissionID \times SchedulableID$

channel $setAltitudeCall : MissionID \times SchedulableID \times \mathbb{R}$

channel $setAltitudeRet : MissionID \times SchedulableID$

channel $setCabinPressureCall : MissionID \times SchedulableID \times \mathbb{R}$

channel $setCabinPressureRet : MissionID \times SchedulableID$

channel $setEmergencyOxygenCall : MissionID \times SchedulableID \times \mathbb{R}$

channel $setEmergencyOxygenRet : MissionID \times SchedulableID$

channel $setFuelRemainingCall : MissionID \times SchedulableID \times \mathbb{R}$

channel $setFuelRemainingRet : MissionID \times SchedulableID$

channel $setHeadingCall : MissionID \times SchedulableID \times \mathbb{R}$

channel $setHeadingRet : MissionID \times SchedulableID$

G.5.2 Schedulables of MainMission

$\textbf{section } \mathit{ACModeChangerApp} \textbf{ parents } \mathit{TopLevelMissionSequencerChan}, \mathit{MissionId},$
$\quad \mathit{MissionIds}, \mathit{SchedulableId}, \mathit{SchedulableIds}, \mathit{ACModeChangerClass}, \mathit{MethodCallBindingChannels}$

$\textbf{process } \mathit{ACModeChangerApp} \mathrel{\widehat=}$
$\quad \mathit{controllingMission} : \mathit{MissionID} \bullet \textbf{begin}$

$$\mathit{GetNextMission} \mathrel{\widehat=} \textbf{var } \mathit{ret} : \mathit{MissionID} \bullet$$
$$\left(\begin{array}{l} \mathit{getNextMissionCall} \,.\, \mathit{ACModeChangerSID} \longrightarrow \\ \mathit{ret} := \mathit{this} \,.\, \mathit{getNextMission}(); \\ \mathit{getNextMissionRet} \,.\, \mathit{ACModeChangerSID} \,!\, \mathit{ret} \longrightarrow \\ \textbf{Skip} \end{array}\right)$$

$$\mathit{Methods} \mathrel{\widehat=}$$
$$\Big(\mathit{GetNextMission}\Big) \;;\; \mathit{Methods}$$

$$\bullet\ (\mathit{Methods}) \mathbin{\triangle} (\mathit{end\_sequencer\_app} \,.\, \mathit{ACModeChangerSID} \longrightarrow \textbf{Skip})$$

$\textbf{end}$

section $ACModeChangerClass$ **parents** $scj\_prelude, SchedulableId, SchedulableIds,$
$SafeletChan, MethodCallBindingChannels, MissionId, MissionIds$

class $ACModeChangerClass \mathrel{\widehat=}$ **begin**

state $State$

$modesLeft : \mathbb{Z}$

state $State$

initial $Init$

$State\,'$

$modesLeft' = 3$

protected $getNextMission \mathrel{\widehat=}$ **var** $ret : MissionID$ $\bullet$

$$\left(\begin{array}{l}
\mathbf{if}\ (modesLeft = 3) \longrightarrow \\
\qquad \left(\begin{array}{l} modesLeft := modesLeft - 1; \\ ret := TakeOffMissionMID \end{array}\right) \\
[\!]\ \neg\ (modesLeft = 3) \longrightarrow \\
\qquad \mathbf{if}\ (modesLeft = 2) \longrightarrow \\
\qquad \left(\begin{array}{l} modesLeft := modesLeft - 1; \\ ret := CruiseMissionMID \end{array}\right) \\
[\!]\ \neg\ (modesLeft = 2) \longrightarrow \\
\qquad \mathbf{if}\ (modesLeft = 1) \longrightarrow \\
\qquad \left(\begin{array}{l} modesLeft := modesLeft - 1; \\ ret := LandMissionMID \end{array}\right) \\
[\!]\ \neg\ (modesLeft = 1) \longrightarrow \\
\qquad \left(ret := nullMissionId\right) \\
\mathbf{fi} \\
\mathbf{fi} \\
\mathbf{fi}
\end{array}\right)$$

$\bullet$ **Skip**

end

section *ControlHandlerApp* **parents** *AperiodicEventHandlerChan*, *SchedulableId*,
SchedulableIds, *MethodCallBindingChannels*

process *ControlHandlerApp* $\widehat{=}$ **begin**

$handleAsyncEvent \mathrel{\widehat{=}}$
$$\left(\begin{array}{l} handleAsyncEventCall\,.\,ControlHandlerSID \longrightarrow \\ handleAsyncEventRet\,.\,ControlHandlerSID \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$Methods \mathrel{\widehat{=}}$
$\left(handleAsyncEvent\right)\ ;\ Methods$

$\bullet\ (Methods) \mathbin{\triangle} (end\_aperiodic\_app\,.\,ControlHandlerSID \longrightarrow \mathbf{Skip})$

end

section *CommunicationsHandlerApp* **parents** *AperiodicEventHandlerChan*,
SchedulableId, *SchedulableIds*, *MethodCallBindingChannels*

process *CommunicationsHandlerApp* $\widehat{=}$ **begin**

$handleAsyncEvent \mathrel{\widehat{=}}$
$$\left(\begin{array}{l} handleAsyncEventCall\,.\,CommunicationsHandlerSID \longrightarrow \\ handleAsyncEventRet\,.\,CommunicationsHandlerSID \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$Methods \mathrel{\widehat{=}}$
$\left(handleAsyncEvent\right)\ ;\ Methods$

$\bullet\ (Methods) \mathbin{\triangle} (end\_aperiodic\_app\,.\,CommunicationsHandlerSID \longrightarrow \mathbf{Skip})$

end

section *EnvironmentMonitorApp* **parents** *PeriodicEventHandlerChan*, *SchedulableId*, *SchedulableIds*, *MethodCallBindingChannels*

process *EnvironmentMonitorApp* $\widehat{=}$ *controllingMission* : *MissionID* $\bullet$ **begin**

$$
\begin{array}{l}
handleAsyncEvent \mathrel{\widehat=} \\
\left(\begin{array}{l}
handleAsyncEventCall \,.\, EnvironmentMonitorSID \longrightarrow \\
\left(\begin{array}{l}
binder\_setCabinPressureCall \,.\, controllingMission \\
\quad .\, EnvironmentMonitorSID \,!\, 0 \longrightarrow \\
binder\_setCabinPressureRet \,.\, controllingMission \\
\quad .\, EnvironmentMonitorSID \longrightarrow \\
binder\_setEmergencyOxygenCall \,.\, controllingMission \\
\quad .\, EnvironmentMonitorSID \,!\, 0 \longrightarrow \\
binder\_setEmergencyOxygenRet \,.\, controllingMission \\
\quad .\, EnvironmentMonitorSID \longrightarrow \\
binder\_setFuelRemainingCall \,.\, controllingMission \\
\quad .\, EnvironmentMonitorSID \,!\, 0 \longrightarrow \\
binder\_setFuelRemainingRet \,.\, controllingMission \\
\quad .\, EnvironmentMonitorSID \longrightarrow \\
\mathbf{Skip}
\end{array}\right) ; \\
handleAsyncEventRet \,.\, EnvironmentMonitorSID \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
Methods \mathrel{\widehat=} \\
\left(handleAsyncEvent\right) ; \ Methods
\end{array}
$$

$\bullet$ $(Methods) \triangle (end\_periodic\_app \,.\, EnvironmentMonitorSID \longrightarrow \mathbf{Skip})$

end

section *FlightSensorsMonitorApp* **parents** *PeriodicEventHandlerChan*, *SchedulableId*, *SchedulableIds*, *MethodCallBindingChannels*

process *FlightSensorsMonitorApp* $\widehat{=}$
controllingMission : *MissionID* • **begin**

$$
\mathit{handleAsyncEvent} \mathrel{\widehat=}
\left(\begin{array}{l}
\mathit{handleAsyncEventCall} \,.\, \mathit{FlightSensorsMonitorSID} \longrightarrow \\
\left(\begin{array}{l}
\mathit{binder\_setAirSpeedCall} \,.\, \mathit{controllingMission} \\
\quad .\, \mathit{FlightSensorsMonitorSID} \,!\, 0 \longrightarrow \\
\mathit{binder\_setAirSpeedRet} \,.\, \mathit{controllingMission} \,.\, \mathit{FlightSensorsMonitorSID} \longrightarrow \\
\mathit{binder\_setAltitudeCall} \,.\, \mathit{controllingMission} \\
\quad .\, \mathit{FlightSensorsMonitorSID} \,!\, 0 \longrightarrow \\
\mathit{binder\_setAltitudeRet} \,.\, \mathit{controllingMission} \,.\, \mathit{FlightSensorsMonitorSID} \longrightarrow \\
\mathit{binder\_setHeadingCall} \,.\, \mathit{controllingMission} \\
\quad .\, \mathit{FlightSensorsMonitorSID} \,!\, 0 \longrightarrow \\
\mathit{binder\_setHeadingRet} \,.\, \mathit{controllingMission} \,.\, \mathit{FlightSensorsMonitorSID} \longrightarrow \\
\mathbf{Skip}
\end{array}\right) ; \\
\mathit{handleAsyncEventRet} \,.\, \mathit{FlightSensorsMonitorSID} \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
$$

$$
\mathit{Methods} \mathrel{\widehat=} \\
\left(\mathit{handleAsyncEvent}\right) ; \ \mathit{Methods}
$$

$$
\bullet\ (\mathit{Methods}) \mathbin{\triangle} (\mathit{end\_periodic\_app} \,.\, \mathit{FlightSensorsMonitorSID} \longrightarrow \mathbf{Skip})
$$

end

G.5.3 TakeOffMission

section *TakeOffMissionApp* **parents** *scj_prelude*, *MissionId*, *MissionIds*, *SchedulableId*, *SchedulableIds*, *MissionChan*, *SchedulableMethChan*, *TakeOffMissionMethChan*, *TakeOffMissionClass*, *MethodCallBindingChannels*, *LandingGearMethChan*

process $TakeOffMissionApp \mathrel{\widehat=}$
$controllingMission : MissionID \bullet$ **begin**

$$\begin{array}{|l}
\hline
\textit{State} \\
\quad this : \mathbf{ref}\ TakeOffMissionClass \\
\hline
\end{array}$$

state *State*

$$\begin{array}{|l}
\hline
\textit{Init} \\
\quad State' \\
\hline
\quad this' = \mathbf{new}\ TakeOffMissionClass() \\
\hline
\end{array}$$

$$InitializePhase \mathrel{\widehat=}$$
$$\left(\begin{array}{l}
initializeCall\,.\,TakeOffMissionMID \longrightarrow \\
register\,!\,LandingGearHandlerSID\,!\,TakeOffMissionMID \longrightarrow \\
register\,!\,TakeOffMonitorSID\,!\,TakeOffMissionMID \longrightarrow \\
register\,!\,TakeOffFailureHandlerSID\,!\,TakeOffMissionMID \longrightarrow \\
initializeRet\,.\,TakeOffMissionMID \longrightarrow \\
\mathbf{Skip}
\end{array}\right)$$

$$CleanupPhase \mathrel{\widehat=} \mathbf{var}\ ret : \mathbb{B} \bullet$$
$$\left(\begin{array}{l}
cleanupMissionCall\,.\,TakeOffMissionMID \longrightarrow \\
\left(\begin{array}{l}
ret := (\neg\ this\,.\,abort); \\
cleanupMissionRet\,.\,TakeOffMissionMID\,!\,ret \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}\right)$$

$$takeOffAbortMeth \mathrel{\widehat=}$$
$$\left(\begin{array}{l}
takeOffAbortCall\,.\,TakeOffMissionMID\,?\,caller \longrightarrow \\
this\,.\,takeOffAbort(); \\
takeOffAbortRet\,.\,TakeOffMissionMID\,.\,caller \longrightarrow \\
\mathbf{Skip}
\end{array}\right)$$

$$deployLandingGearMeth \mathrel{\widehat=} \left(\begin{array}{l} deployLandingGearCall \,.\, TakeOffMissionMID \,?\, caller \longrightarrow \\ this \,.\, deployLandingGear(); \\ deployLandingGearRet \,.\, TakeOffMissionMID \,.\, caller \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$stowLandingGearMeth \mathrel{\widehat=} \left(\begin{array}{l} stowLandingGearCall \,.\, TakeOffMissionMID \,?\, caller \longrightarrow \\ this \,.\, stowLandingGear(); \\ stowLandingGearRet \,.\, TakeOffMissionMID \,.\, caller \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$isLandingGearDeployedMeth \mathrel{\widehat=} \mathbf{var}\; ret : \mathbb{B} \bullet \left(\begin{array}{l} isLandingGearDeployedCall \,.\, TakeOffMissionMID \,?\, caller \longrightarrow \\ \left(\begin{array}{l} ret := this \,.\, isLandingGearDeployed(); \\ ;\; isLandingGearDeployedRet \,.\, TakeOffMissionMID \,.\, caller \,!\, ret \longrightarrow \\ \mathbf{Skip} \end{array}\right) \end{array}\right)$$

$$Methods \mathrel{\widehat=} \left(\begin{array}{l} InitializePhase \\ \Box \\ CleanupPhase \\ \Box \\ takeOffAbortMeth \\ \Box \\ deployLandingGearMeth \\ \Box \\ stowLandingGearMeth \\ \Box \\ isLandingGearDeployedMeth \end{array}\right) ;\; Methods$$

$$\bullet\; (Init \;;\; Methods) \mathbin{\triangle} (end\_mission\_app \,.\, TakeOffMissionMID \longrightarrow \mathbf{Skip})$$

end

section $\mathit{TakeOffMissionClass}$ **parents** $\mathit{scj\_prelude}, \mathit{SchedulableId}, \mathit{SchedulableIds},$
$\mathit{SafeletChan}, \mathit{MethodCallBindingChannels}$

class $\mathit{TakeOffMissionClass} \mathrel{\widehat=}$ **begin**

state State

$\mathit{SAFE\_AIRSPEED\_THRESHOLD} : \mathbb{R}$
$\mathit{TAKEOFF\_ALTITUDE} : \mathbb{R}$
$\mathit{abort} : \mathbb{B}$
$\mathit{landingGearDeployed} : \mathbb{B}$

state State

initial Init

State'

$\mathit{SAFE\_AIRSPEED\_THRESHOLD}' = 10.0$
$\mathit{TAKEOFF\_ALTITUDE}' = 10.0$
$\mathit{abort}' = \mathit{false}$

public $\mathit{takeOffAbort} \mathrel{\widehat=}$
$\left(\mathit{abort} := \mathbf{True}\right)$

public $\mathit{deployLandingGear} \mathrel{\widehat=}$
$\left(\mathit{landingGearDeployed} := \mathbf{True}\right)$

public $\mathit{stowLandingGear} \mathrel{\widehat=}$
$\left(\mathit{landingGearDeployed} := \mathbf{False}\right)$

public $\mathit{isLandingGearDeployed} \mathrel{\widehat=}$
$\left(\mathit{ret} := \mathit{landingGearDeployed}\right)$

• **Skip**

end

section $\mathit{TakeOffMissionMethChan}$ **parents** $\mathit{GlobalTypes}, \mathit{MissionId}, \mathit{SchedulableId}$

channel $\mathit{takeOffAbortCall} : \mathit{MissionID} \times \mathit{SchedulableID}$
channel $\mathit{takeOffAbortRet} : \mathit{MissionID} \times \mathit{SchedulableID}$

G.5.4 Schedulables of TakeOffMission

section $\mathit{LandingGearMethChan}$ **parents** $\mathit{GlobalTypes}, \mathit{MissionId}, \mathit{SchedulableId}$

channel $\mathit{deployLandingGearCall} : \mathit{MissionID} \times \mathit{SchedulableID}$
channel $\mathit{deployLandingGearRet} : \mathit{MissionID} \times \mathit{SchedulableID}$
channel $\mathit{stowLandingGearCall} : \mathit{MissionID} \times \mathit{SchedulableID}$
channel $\mathit{stowLandingGearRet} : \mathit{MissionID} \times \mathit{SchedulableID}$
channel $\mathit{isLandingGearDeployedCall} : \mathit{MissionID} \times \mathit{SchedulableID}$
channel $\mathit{isLandingGearDeployedRet} : \mathit{MissionID} \times \mathit{SchedulableID} \times \mathbb{B}$

section *LandingGearHandlerTakeOffApp* **parents** *AperiodicEventHandlerChan*, *SchedulableId*, *SchedulableIds*, *MethodCallBindingChannels*

process *LandingGearHandlerApp* $\widehat{=}$
  $mission : MissionID \bullet$ **begin**

$handleAsyncEvent \mathrel{\widehat{=}}$

$$\left(\begin{array}{l}
handleAsyncEventCall . LandingGearHandlerSID \longrightarrow \\
\left(\begin{array}{l}
binder\_isLandingGearDeployedCall . mission . LandingGearHandlerSID \longrightarrow \\
binder\_isLandingGearDeployedRet . mission . LandingGearHandlerSID \\
\quad ?\, isLandingGearDeployed \longrightarrow \\
\mathbf{var}\ landingGearIsDeployed : \mathbb{B} \bullet \\
\quad landingGearIsDeployed := isLandingGearDeployed; \\
\mathbf{if}\ landingGearIsDeployed = \mathbf{True} \longrightarrow \\
\quad \left(\begin{array}{l}
binder\_stowLandingGearCall . mission . LandingGearHandlerSID \longrightarrow \\
binder\_stowLandingGearRet . mission . LandingGearHandlerSID \longrightarrow \\
\mathbf{Skip}
\end{array}\right) \\
[\!]\ \neg\ landingGearIsDeployed = \mathbf{True} \longrightarrow \\
\quad \left(\begin{array}{l}
binder\_deployLandingGearCall . mission . LandingGearHandlerSID \longrightarrow \\
binder\_deployLandingGearRet . mission . LandingGearHandlerSID \longrightarrow \\
\mathbf{Skip}
\end{array}\right) \\
\mathbf{fi}
\end{array}\right); \\
handleAsyncEventRet . LandingGearHandlerSID \longrightarrow \\
\mathbf{Skip}
\end{array}\right)$$

$Methods \mathrel{\widehat{=}}$
$\left(handleAsyncEvent\right) ;\ Methods$

$\bullet\ (Methods) \triangle (end\_aperiodic\_app . LandingGearHandlerSID \longrightarrow \mathbf{Skip})$

end

section *TakeOffFailureHandlerApp* **parents** *AperiodicEventHandlerChan*, *SchedulableId*, *SchedulableIds*, *MethodCallBindingChannels*, *MainMissionMethChan*, *MissionMethChan*

process *TakeOffFailureHandlerApp* $\widehat{=}$
mainMission : *MissionID*;
takeoffMission : *MissionID*;
threshold : $\mathbb{R}$ $\bullet$ **begin**

$$
\begin{array}{l}
handleAsyncEvent \mathrel{\widehat=} \\
\left(\begin{array}{l}
handleAsyncEventCall\,.\,TakeOffFailureHandlerSID \longrightarrow \\
\left(\begin{array}{l}
binder\_getAirSpeedCall\,.\,mainMission \\
\qquad .\,TakeOffFailureHandlerSID \longrightarrow \\
binder\_getAirSpeedRet\,.\,mainMission \\
\qquad .\,TakeOffFailureHandlerSID\,?\,getAirSpeed \longrightarrow \\
\mathbf{var}\ currentSpeed : \mathbb{R} \bullet currentSpeed := getAirSpeed; \\
\mathbf{if}\ (currentSpeed < threshold) \longrightarrow \\
\quad \left(\begin{array}{l}
binder\_takeOffAbortCall\,.\,takeoffMission \\
\qquad .\,TakeOffFailureHandlerSID \longrightarrow \\
binder\_takeOffAbortRet\,.\,takeoffMission \\
\qquad .\,TakeOffFailureHandlerSID \longrightarrow \\
requestTerminationCall\,.\,takeoffMission \\
\qquad .\,TakeOffFailureHandlerSID \longrightarrow \\
requestTerminationRet\,.\,takeoffMission \\
\qquad .\,TakeOffFailureHandlerSID\,?\,rt \longrightarrow \\
\mathbf{Skip}
\end{array}\right) \\
[\!]\ \neg\ (currentSpeed < threshold) \longrightarrow \\
\quad \mathbf{Skip} \\
\mathbf{fi}
\end{array}\right); \\
handleAsyncEventRet\,.\,TakeOffFailureHandlerSID \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
Methods \mathrel{\widehat=} \\
\left(handleAsyncEvent\right);\ Methods
\end{array}
$$

$\bullet$ $(Methods) \mathbin{\triangle} (end\_aperiodic\_app\,.\,TakeOffFailureHandlerSID \longrightarrow \mathbf{Skip})$

end

section *TakeOffMonitorApp* **parents** *PeriodicEventHandlerChan*, *SchedulableId*, *SchedulableIds*, *MethodCallBindingChannels*, *MainMissionMethChan*, *MissionMethChan*

process *TakeOffMonitorApp* $\widehat{=}$
$mainMission : MissionID;$
$takeOffMission : MissionID;$
$takeOffAltitude : \mathbb{R};$
$landingGearHandler : SchedulableID$ ● **begin**

$$handleAsyncEvent \mathrel{\widehat{=}}$$
$$\left(\begin{array}{l}
handleAsyncEventCall \,.\, TakeOffMonitorSID \longrightarrow \\
\left(\begin{array}{l}
binder\_getAltitudeCall \,.\, mainMission \,.\, TakeOffMonitorSID \longrightarrow \\
binder\_getAltitudeRet \,.\, mainMission \,.\, TakeOffMonitorSID \,?\, getAltitude \longrightarrow \\
\mathbf{var}\ altitude : \mathbb{R} \bullet altitude := getAltitude; \\
\mathbf{if}\ (altitude > takeOffAltitude) \longrightarrow \\
\quad \left(\begin{array}{l}
release \,.\, landingGearHandler \longrightarrow \\
requestTerminationCall \,.\, takeOffMission \,.\, TakeOffMonitorSID \longrightarrow \\
requestTerminationRet \,.\, takeOffMission \,.\, TakeOffMonitorSID \,?\, rt \longrightarrow \\
\mathbf{Skip}
\end{array}\right) \\
[\!]\ \neg\ (altitude > takeOffAltitude) \longrightarrow \mathbf{Skip} \\
\mathbf{fi}
\end{array}\right); \\
handleAsyncEventRet \,.\, TakeOffMonitorSID \longrightarrow \\
\mathbf{Skip}
\end{array}\right)$$

$$Methods \mathrel{\widehat{=}}$$
$$\left(handleAsyncEvent\right) \; ; \; Methods$$

$\bullet\ (Methods) \mathbin{\triangle} (end\_periodic\_app \,.\, TakeOffMonitorSID \longrightarrow \mathbf{Skip})$

end

G.5.5 CruiseMission

section *CruiseMissionApp* **parents** *scj_prelude*, *MissionId*, *MissionIds*, *SchedulableId*, *SchedulableIds*, *MissionChan*, *SchedulableMethChan*, *MethodCallBindingChannels*

process *CruiseMissionApp* $\widehat{=}$
controllingMission : *MissionID* $\bullet$ **begin**

$$InitializePhase \mathrel{\widehat{=}} \left(\begin{array}{l} initializeCall \,.\, CruiseMissionMID \longrightarrow \\ register \,!\, BeginLandingHandlerSID \,!\, CruiseMissionMID \longrightarrow \\ register \,!\, NavigationMonitorSID \,!\, CruiseMissionMID \longrightarrow \\ initializeRet \,.\, CruiseMissionMID \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$CleanupPhase \mathrel{\widehat{=}} \left(\begin{array}{l} cleanupMissionCall \,.\, CruiseMissionMID \longrightarrow \\ cleanupMissionRet \,.\, CruiseMissionMID \,!\, \mathbf{True} \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$Methods \mathrel{\widehat{=}} \left(\begin{array}{l} InitializePhase \\ \Box \\ CleanupPhase \end{array}\right) ; \; Methods$$

$$\bullet \; (Methods) \vartriangle (end\_mission\_app \,.\, CruiseMissionMID \longrightarrow \mathbf{Skip})$$

end

G.5.6 Schedulables of CruiseMission

section $\mathit{BeginLandingHandlerApp}$ **parents** $\mathit{AperiodicEventHandlerChan}$, $\mathit{SchedulableId}, \mathit{SchedulableIds}, \mathit{MethodCallBindingChannels}, \mathit{MissionMethChan}$

process $\mathit{BeginLandingHandlerApp} \mathrel{\widehat=}$
$\mathit{controllingMission} : \mathit{MissionID} \bullet$ **begin**

$$
\mathit{handleAsyncEvent} \mathrel{\widehat=}
\left(\begin{array}{l}
\mathit{handleAsyncEventCall} \,.\, \mathit{BeginLandingHandlerSID} \longrightarrow \\
\left(\begin{array}{l}
\mathit{requestTerminationCall} \,.\, \mathit{controllingMission} \,.\, \mathit{BeginLandingHandlerSID} \longrightarrow \\
\mathit{requestTerminationRet} \,.\, \mathit{controllingMission} \,.\, \mathit{BeginLandingHandlerSID} \,?\, rt \longrightarrow \\
\mathbf{Skip}
\end{array}\right) ; \\
\mathit{handleAsyncEventRet} \,.\, \mathit{BeginLandingHandlerSID} \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
$$

$$
\mathit{Methods} \mathrel{\widehat=}
\Big(\mathit{handleAsyncEvent}\Big) ;\ \mathit{Methods}
$$

$\bullet\ (\mathit{Methods}) \triangle (\mathit{end\_aperiodic\_app} \,.\, \mathit{BeginLandingHandlerSID} \longrightarrow \mathbf{Skip})$

end

section *NavigationMonitorApp* **parents** *PeriodicEventHandlerChan*, *SchedulableId*, *SchedulableIds*, *MethodCallBindingChannels*, *MainMissionMethChan*

process *NavigationMonitorApp* $\widehat{=}$ *mainMission* : *MissionID* $\bullet$ **begin**

$$
\mathit{handleAsyncEvent} \mathrel{\widehat{=}}
\left(\begin{array}{l}
\mathit{handleAsyncEventCall} \,.\, \mathit{NavigationMonitorSID} \longrightarrow \\
\left(\begin{array}{l}
\mathit{binder\_getHeadingCall} \,.\, \mathit{mainMission} \,.\, \mathit{NavigationMonitorSID} \longrightarrow \\
\mathit{binder\_getHeadingRet} \,.\, \mathit{mainMission} \\
\qquad .\, \mathit{NavigationMonitorSID} \,?\, \mathit{getHeading} \longrightarrow \\
\mathbf{var}\ \mathit{heading} : \mathbb{R} \bullet \mathit{heading} := \mathit{getHeading}; \\
\mathit{binder\_getAirSpeedCall} \,.\, \mathit{mainMission} \,.\, \mathit{NavigationMonitorSID} \longrightarrow \\
\mathit{binder\_getAirSpeedRet} \,.\, \mathit{mainMission} \\
\qquad .\, \mathit{NavigationMonitorSID} \,?\, \mathit{getAirSpeed} \longrightarrow \\
\mathbf{var}\ \mathit{airSpeed} : \mathbb{R} \bullet \mathit{airSpeed} := \mathit{getAirSpeed}; \\
\mathit{binder\_getAltitudeCall} \,.\, \mathit{mainMission} \,.\, \mathit{NavigationMonitorSID} \longrightarrow \\
\mathit{binder\_getAltitudeRet} \,.\, \mathit{mainMission} \\
\qquad .\, \mathit{NavigationMonitorSID} \,?\, \mathit{getAltitude} \longrightarrow \\
\mathbf{var}\ \mathit{altitude} : \mathbb{R} \bullet \mathit{altitude} := \mathit{getAltitude}
\end{array}\right); \\
\mathit{handleAsyncEventRet} \,.\, \mathit{NavigationMonitorSID} \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
$$

$$
\mathit{Methods} \mathrel{\widehat{=}} \left(\mathit{handleAsyncEvent}\right) ; \ \mathit{Methods}
$$

$$
\bullet\ (\mathit{Methods}) \vartriangle (\mathit{end\_periodic\_app} \,.\, \mathit{NavigationMonitorSID} \longrightarrow \mathbf{Skip})
$$

end

G.5.7 LandMission

section $\mathit{LandMissionApp}$ **parents** $\mathit{scj\_prelude}, \mathit{MissionId}, \mathit{MissionIds}, \mathit{SchedulableId}, \mathit{SchedulableIds}, \mathit{MissionChan}, \mathit{SchedulableMethChan}, \mathit{LandingGearMethChan}, \mathit{LandMissionClass}, \mathit{MethodCallBindingChannels}$

process $\mathit{LandMissionApp} \mathrel{\widehat=}$
$\mathit{controllingMission} : \mathit{MissionID} \bullet$ **begin**

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{State}} \\ \hline
\mathit{this} : \mathbf{ref}\ \mathit{LandMissionClass} \\ \hline
\end{array}$$

state State

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{Init}} \\ \hline
\mathit{State}' \\ \hline
\mathit{this}' = \mathbf{new}\ \mathit{LandMissionClass}() \\ \hline
\end{array}$$

$$\mathit{InitializePhase} \mathrel{\widehat=}$$
$$\left(\begin{array}{l}
\mathit{initializeCall}\,.\,\mathit{LandMissionMID} \longrightarrow \\
\mathit{register}\,!\,\mathit{GroundDistanceMonitorSID}\,!\,\mathit{LandMissionMID} \longrightarrow \\
\mathit{register}\,!\,\mathit{LandingGearHandlerLandSID}\,!\,\mathit{LandMissionMID} \longrightarrow \\
\mathit{register}\,!\,\mathit{InstrumentLandingSystemMonitorSID}\,!\,\mathit{LandMissionMID} \longrightarrow \\
\mathit{register}\,!\,\mathit{SafeLandingHandlerSID}\,!\,\mathit{LandMissionMID} \longrightarrow \\
\mathit{initializeRet}\,.\,\mathit{LandMissionMID} \longrightarrow \\
\mathbf{Skip}
\end{array}\right)$$

$$\mathit{CleanupPhase} \mathrel{\widehat=} \mathbf{var}\ \mathit{ret} : \mathbb{B} \bullet$$
$$\left(\begin{array}{l}
\mathit{cleanupMissionCall}\,.\,\mathit{LandMissionMID} \longrightarrow \\
\mathit{ret} := \mathbf{False}; \\
\mathit{cleanupMissionRet}\,.\,\mathit{LandMissionMID}\,!\,\mathit{ret} \longrightarrow \\
\mathbf{Skip}
\end{array}\right)$$

$$\mathit{deployLandingGearMeth} \mathrel{\widehat=}$$
$$\left(\begin{array}{l}
\mathit{deployLandingGearCall}\,.\,\mathit{LandMissionMID}\,?\,\mathit{caller} \longrightarrow \\
\mathit{this}\,.\,\mathit{deployLandingGear}(); \\
\mathit{deployLandingGearRet}\,.\,\mathit{LandMissionMID}\,.\,\mathit{caller} \longrightarrow \\
\mathbf{Skip}
\end{array}\right)$$

$$stowLandingGearMeth \mathrel{\widehat=}$$
$$\left(\begin{array}{l} stowLandingGearCall \,.\, LandMissionMID \,?\, caller \longrightarrow \\ this \,.\, stowLandingGear(); \\ stowLandingGearRet \,.\, LandMissionMID \,.\, caller \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$isLandingGearDeployedMeth \mathrel{\widehat=} \mathbf{var}\; ret : \mathbb{B} \bullet$$
$$\left(\begin{array}{l} isLandingGearDeployedCall \,.\, LandMissionMID \,?\, caller \longrightarrow \\ ret := this \,.\, isLandingGearDeployed(); \\ isLandingGearDeployedRet \,.\, LandMissionMID \,.\, caller \,!\, ret \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$$Methods \mathrel{\widehat=} \left(\begin{array}{l} InitializePhase \\ \Box \\ CleanupPhase \\ \Box \\ deployLandingGearMeth \\ \Box \\ stowLandingGearMeth \\ \Box \\ isLandingGearDeployedMeth \end{array}\right) ;\; Methods$$

$$\bullet\ (Methods) \mathbin{\triangle} (end\_mission\_app \,.\, LandMissionMID \longrightarrow \mathbf{Skip})$$

end

section $\mathit{LandMissionClass}$ **parents** $\mathit{scj\_prelude}, \mathit{SchedulableId}, \mathit{SchedulableIds},$
$\mathit{SafeletChan}, \mathit{MethodCallBindingChannels}$

class $\mathit{LandMissionClass} \mathrel{\widehat=}$ **begin**

state State

$\mathit{SAFE\_LANDING\_ALTITUDE} : \mathbb{R}$
$\mathit{ALTITUDE\_READING\_ON\_GROUND} : \mathbb{R}$
$\mathit{abort} : \mathbb{B}$
$\mathit{landingGearDeployed} : \mathbb{B}$

state State

initial Init

State'

$\mathit{SAFE\_LANDING\_ALTITUDE}' = 10.0$
$\mathit{ALTITUDE\_READING\_ON\_GROUND}' = 0.0$
$\mathit{abort}' = \mathit{false}$

public $\mathit{deployLandingGear} \mathrel{\widehat=}$
$\left(\mathit{landingGearDeployed} := \mathbf{True}\right)$

public $\mathit{stowLandingGear} \mathrel{\widehat=}$
$\left(\mathit{landingGearDeployed} := \mathbf{False}\right)$

public $\mathit{isLandingGearDeployed} \mathrel{\widehat=}$
$\left(\mathit{ret} := \mathit{landingGearDeployed}\right)$

• **Skip**

end

G.5.8 Schedulables of LandMission

section *LandingGearHandlerLandApp* **parents** *AperiodicEventHandlerChan*, *SchedulableId*, *SchedulableIds*, *MethodCallBindingChannels*, *LandMissionMethChan*

process *LandingGearHandlerLandApp* $\widehat{=}$
mission : *MissionID* • **begin**

$$
handleAsyncEvent \mathrel{\widehat=} \\
\left(\begin{array}{l}
handleAsyncEventCall \,.\, LandingGearHandlerLandSID \longrightarrow \\
\left(\begin{array}{l}
binder\_isLandingGearDeployedCall \,.\, mission \,.\, LandingGearHandlerLandSID \longrightarrow \\
binder\_isLandingGearDeployedRet \,.\, mission \,.\, LandingGearHandlerLandSID \\
\qquad ?\, isLandingGearDeployed \longrightarrow \\
\mathbf{var}\ landingGearIsDeployed : \mathbb{B} \bullet \\
\qquad landingGearIsDeployed := isLandingGearDeployed; \\
\mathbf{if}\ landingGearIsDeployed \longrightarrow \\
\qquad \left(\begin{array}{l}
binder\_stowLandingGearCall \,.\, mission \,.\, LandingGearHandlerLandSID \longrightarrow \\
binder\_stowLandingGearRet \,.\, mission \,.\, LandingGearHandlerLandSID \longrightarrow \\
\mathbf{Skip}
\end{array}\right) \\
[\!] \neg\ landingGearIsDeployed \longrightarrow \\
\qquad \left(\begin{array}{l}
binder\_deployLandingGearCall \,.\, mission \,.\, LandingGearHandlerLandSID \longrightarrow \\
binder\_deployLandingGearRet \,.\, mission \,.\, LandingGearHandlerLandSID \longrightarrow \\
\mathbf{Skip}
\end{array}\right) \\
\mathbf{fi}
\end{array}\right); \\
handleAsyncEventRet \,.\, LandingGearHandlerLandSID \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
$$

$$
Methods \mathrel{\widehat=} \\
\left(handleAsyncEvent\right) ;\ Methods
$$

$\bullet\ (Methods) \vartriangle (end\_aperiodic\_app \,.\, LandingGearHandlerLandSID \longrightarrow \mathbf{Skip})$

end

section *SafeLandingHandlerApp* **parents** *AperiodicEventHandlerChan*,
SchedulableId, *SchedulableIds*, *MethodCallBindingChannels*, *MainMissionMethChan*

process *SafeLandingHandlerApp* $\widehat{=}$
mainMission : *MissionID*;
threshold : $\mathbb{R}$ ● **begin**

$$
handleAsyncEvent \mathrel{\widehat=}
\left(\begin{array}{l}
handleAsyncEventCall \,.\, SafeLandingHandlerSID \longrightarrow \\
\left(\begin{array}{l}
binder\_getAltitudeCall \,.\, mainMission \,.\, SafeLandingHandlerSID \longrightarrow \\
binder\_getAltitudeRet \,.\, mainMission \,.\, SafeLandingHandlerSID \\
\qquad ?\, getAltitude \longrightarrow \\
\mathbf{var}\ altitude : \mathbb{R} \bullet altitude := getAltitude; \\
\mathbf{if}\ (altitude < threshold) \longrightarrow \\
\quad \mathbf{Skip} \\
[\!]\ \neg\ (altitude < threshold) \longrightarrow \\
\quad \mathbf{Skip} \\
\mathbf{fi}
\end{array}\right) ; \\
handleAsyncEventRet \,.\, SafeLandingHandlerSID \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
$$

$$
Methods \mathrel{\widehat=}
\left(handleAsyncEvent\right) ;\ Methods
$$

$\bullet\ (Methods) \vartriangle (end\_aperiodic\_app \,.\, SafeLandingHandlerSID \longrightarrow \mathbf{Skip})$

end

section *GroundDistanceMonitorApp* **parents** *PeriodicEventHandlerChan*,
SchedulableId, *SchedulableIds*, *MethodCallBindingChannels*, *MissionMethChan*

process *GroundDistanceMonitorApp* $\widehat{=}$
mainMission : *MissionID*;
readingOnGround : $\mathbb{R}$ ● **begin**

$$
handleAsyncEvent \mathrel{\widehat{=}}
\left(\begin{array}{l}
handleAsyncEventCall \,.\, GroundDistanceMonitorSID \longrightarrow \\
\left(\begin{array}{l}
binder\_getAltitudeCall \,.\, mainMission \,.\, GroundDistanceMonitorSID \longrightarrow \\
binder\_getAltitudeRet \,.\, mainMission \,.\, GroundDistanceMonitorSID \,?\, getAltitude \longrightarrow \\
\mathbf{var}\ distance : \mathbb{R} \bullet distance := getAltitude; \\
\mathbf{if}\ (distance = readingOnGround) \longrightarrow \\
\quad \left(\begin{array}{l}
requestTerminationCall \,.\, mainMission \,.\, GroundDistanceMonitorSID \longrightarrow \\
requestTerminationRet \,.\, mainMission \,.\, GroundDistanceMonitorSID \,?\, rt \longrightarrow \\
\mathbf{Skip}
\end{array}\right) \\
[\!] \neg\, (distance = readingOnGround) \longrightarrow \mathbf{Skip} \\
\mathbf{fi}
\end{array}\right); \\
handleAsyncEventRet \,.\, GroundDistanceMonitorSID \longrightarrow \\
\mathbf{Skip}
\end{array}\right)
$$

$$
Methods \mathrel{\widehat{=}} \\
\left(handleAsyncEvent\right) \;;\; Methods
$$

$$
\bullet\ (Methods) \mathbin{\triangle} (end\_periodic\_app \,.\, GroundDistanceMonitorSID \longrightarrow \mathbf{Skip})
$$

end

section *InstrumentLandingSystemMonitorApp* **parents** *PeriodicEventHandlerChan*, *SchedulableId*, *SchedulableIds*, *MethodCallBindingChannels*

process *InstrumentLandingSystemMonitorApp* $\widehat{=}$
$mission : MissionID$ $\bullet$ **begin**

$handleAsyncEvent \mathrel{\widehat{=}}$
$$\left(\begin{array}{l} handleAsyncEventCall \,.\, InstrumentLandingSystemMonitorSID \longrightarrow \\ handleAsyncEventRet \,.\, InstrumentLandingSystemMonitorSID \longrightarrow \\ \mathbf{Skip} \end{array}\right)$$

$Methods \mathrel{\widehat{=}}$
$$\left(handleAsyncEvent\right) \;;\; Methods$$

$$\bullet\ (Methods) \triangle (end\_periodic\_app \,.\, InstrumentLandingSystemMonitorSID \longrightarrow \mathbf{Skip})$$

end

Bibliography

[1] Abrial, J.R., Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press (3 Nov 2005)

[2] Abrial, J.R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: an Open Toolset for Modelling and Reasoning in Event-B. Int. J. Softw. Tools Technol. Trans. 12(6), 447–466 (7 Apr 2010)

[3] Aicas: JamiacaVM. `http://www.aicas.com/jamaica.html` (2013)

[4] Andersen, J.L., Todberg, M., Dalsgaard, A.E., Hansen, R.R.: Worst-Case Memory Consumption Analysis for SCJ. In: Proceedings of the 11th International Workshop on Java Technologies for Real-time and Embedded Systems. pp. 2–10. JTRES '13, ACM, New York, NY, USA (2013)

[5] Beg, A., Butterfield, A.: Development of a Prototype Translator from Circus to CSPm. In: 2015 International Conference on Open Source Systems Technologies (ICOSST). pp. 16–23 (Dec 2015)

[6] Beg, A., Butterfield, A.: Linking a State-Rich Process Algebra to a State-Free Algebra to Verify Software/Hardware Implementation. In: Proceedings of the 8th International Conference on Frontiers of Information Technology. p. 47. ACM (21 Dec 2010)

[7] Bogholm, T., Thomsen, B., Larsen, K.G., Mycroft, A.: Schedulability Analysis Abstractions for Safety Critical Java. In: Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2012 IEEE 15th International Symposium on. pp. 71–78 (Apr 2012)

[8] Bollella, G., Brosgol, B., Gosling, J., Dibble, P., Furr, S., Turnbull, M.: The Real-Time Specification for Java. Addison Wesley Longman (2000)

[9] Brosgol, B.M., Wellings, A.J.: A Comparison of Ada and Real-Time Java for Safety-Critical Applications. In: Pinho, L., González Harbour, M. (eds.) Reliable Software Technologies – Ada-Europe 2006, Lecture Notes in Computer Science, vol. 4006, pp. 13–26. Springer Berlin Heidelberg (5 Jun 2006)

[10] Bubel, R., Montoya, A.F., Hähnle, R.: Analysis of Executable Software Models. In: Formal Methods for Executable Software Models, pp. 1–25. Lecture Notes in Computer Science, Springer International Publishing (16 Jun 2014)

[11] Burmyakov, A., Bini, E., Tovar, E.: The Generalized Multiprocessor Periodic Resource Interface Model for Hierarchical Multiprocessor Scheduling. In: Proceedings of the 20th International Conference on Real-Time and Network Systems. pp. 131–139 (2012)

[12] Burns, A., Wellings, A.J.: Real-Time Systems and Programming Languages: Ada 95, Real-Time Java, and Real-Time POSIX. Addison Wesley (2009)

[13] Burns, A., Wellings, A.J.: Processing Group Parameters in the Real-Time Specification for Java. In: On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops. pp. 360–370 (2003)

[14] Carré, B., Jennings, T.: SPARK: The SPADE Ada Kernal. Tech. rep., Department of Electronic and Computer Sciecne, University of Southampton (Mar 1988)

[15] Carré, B., Garnsworthy, J.: SPARK – An Annotated Ada Subset for Safety-Critical Programming. In: Proceedings of the conference on TRI-ADA '90. pp. 392–402. ACM, Baltimore, Maryland, United States (1990)

[16] Cavalcanti, A., Wellings, A.J., Woodcock, J.: The Safety-Critical Java Memory Model Formalised. Formal Aspects of Computing 25(1), 37–57 (29 Jun 2012)

[17] Cavalcanti, A., Sampaio, A., Woodcock, J.: A Refinement Strategy for Circus. Form. Asp. Comput. 15(2-3), 146–181 (Nov 2003)

[18] Cavalcanti, A., Sampaio, A., Woodcock, J.: Unifying Classes and Processes. Software & Systems Modeling 4(3), 277–296 (7 Jun 2005)

[19] Cavalcanti, A., Wellings, A.J., Woodcock, J., Wei, K., Zeyda, F.: Safety-Critical Java in Circus. In: Proceedings of the 9th International Workshop on Java Technologies for Real-Time and Embedded Systems. pp. 20–29. JTRES '11, ACM, New York, NY, USA (26 Sep 2011)

[20] Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Quesada, J.F.: Maude: Specification and Programming in Rewriting Logic. Theor. Comput. Sci. 285(2), 187–243 (28 Aug 2002)

[21] Corsaro, A., Santoro, C.: Design Patterns for RTSJ Application Development. In: Meersman, R., Tari, Z., Corsaro, A. (eds.) On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops, Lecture Notes in Computer Science, vol. 3292, pp. 394–405. Springer Berlin Heidelberg (25 Oct 2004)

[22] Craigen, D., Saaltink, M., Michell, S.: Ada95 and Critical Systems: An analytical Approach. In: Strohmeier, A. (ed.) Reliable Software Technologies — Ada-Europe '96, Lecture Notes in Computer Science, vol. 1088, pp. 171–182. Springer Berlin / Heidelberg (10 Jun 1996)

[23] Cullyer, W.J., Goodenough, S.J., Wichmann, B.A.: The Choice of Computer Languages for use in Safety-Critical Systems. Software Engineering Journal 6(2), 51–58 (Mar 1991)

[24] Dalsgaard, A.E., Hansen, R.R., Schoeberl, M.: Private Memory Allocation Analysis for Safety-Critical Java. In: Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems. pp. 9–17. JTRES '12, ACM, New York, NY, USA (24 Oct 2012)

[25] Davis, R., Burns, A.: An Investigation into Server Parameter Selection for Hierarchical Fixed Priority Pre-Emptive Systems. In: 16th International Conference on Real-Time and Network Systems (RTNS 2008). pp. 19–28 (2008)

[26] Davis, R.I., Burns, A.: Hierarchical Fixed Priority Pre-Emptive Scheduling. In: Real-Time Systems Symposium, 2005. RTSS 2005. 26th IEEE International. pp. 10–pp (2005)

[27] Davis, R.I., Burns, A.: A Survey of Hard Real-Time Scheduling for Multiprocessor Systems. ACM Computing Surveys (CSUR) 43(4), 1–44 (2011)

[28] Dobbing, B., Burns, A.: The ravenscar tasking profile for high integrity Real-Time programs. Ada Lett. XVIII(6), 1–6 (1998)

[29] EUROCAE and RTCA: Software Considerations in Airborne Systems and Equipment Certification. Tech. Rep. DO-178B/ED-12B (1992)

[30] EUROCAE and RTCA: Software Considerations in Airborne Systems and Equipment Certification. Tech. Rep. DO-178C/ED-12C, EUROCAE and RTCA (2012)

[31] Freitas, A., Cavalcanti, A.: Automatic Translation from Circus to Java. In: FM 2006: Formal Methods, pp. 115–130. Lecture Notes in Computer Science, Springer Berlin Heidelberg (21 Aug 2006)

[32] Frost, C., Jensen, C.S., Luckow, K.S., Thomsen, B.: WCET Analysis of Java Bytecode Featuring Common Execution Environments. In: Proceedings of the 9th International Workshop on Java Technologies for Real-Time and Embedded Systems. pp. 30–39. JTRES '11, ACM, New York, NY, USA (2011)

[33] Gibson-Robinson, T., Armstrong, P., Boulgakov, A., Roscoe, A.W.: FDR3 — A Modern Refinement Checker for CSP. In: Ábrahám, E., Havelund, K. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science, vol. 8413, pp. 187–201 (2014)

[34] Gosling, J., Joy, B., Steele, G.L., Bracha, G., Buckley, A.: The Java Language Specification, Java SE 8 Edition. Addison-Wesley Professional, 1st edn. (2014)

[35] Haddad, G., Hussain, F., Leavens, G.T.: The Design of SafeJML, a Specification Language for SCJ with Support for WCET Specification. In: Proceedings of the 8th International Workshop on Java Technologies for Real-Time and Embedded Systems. pp. 155–163. JTRES '10, ACM, New York, NY, USA (19 Aug 2010)

[36] Halang, W.A., Zalewski, J.: Programming Languages for use in Safety-Related Applications. Annu. Rev. Control 27(1), 39–45 (2003)

[37] Hatton, L.: Safer C: Developing Software for in High-Integrity and Safety-Critical Systems. McGraw-Hill, Inc. (1995)

[38] Havelund, K., Pressburger, T.: Model Checking Java Programs using Java PathFinder. Int. J. Softw. Tools Technol. Trans. 2(4), 366–381 (2000)

[39] Havelund, K., Roşu, G.: Monitoring java programs with java PathExplorer. Electron. Notes Theor. Comput. Sci. 55(2), 200–217 (Oct 2001)

[40] Henties, T., Hunt, J.J., Locke, D., Nilsen, K., Schoeberl, M., Vitek, J.: Java for Safety-Critical Applications. In: 2nd International Workshop on the Certification of Safety-Critical Software Controlled Systems (SafeCert 2009) (2009)

[41] Hunt, J., Long, F.: Java's Reliability: an Analysis of Software Defects in Java. Software, IEEE Proceedings 145(2), 41–50 (1 Dec 1998)

[42] Hunt, J.J.: Realtime Java Technology in Avionics Systems. In: Proceedings of the 8th International Workshop on Java Technologies for Real-Time and Embedded Systems. pp. 138–147. JTRES '10, ACM, New York, NY, USA (19 Aug 2010)

[43] Hunt, J.J., Nilsen, K.: Safety-Critical Java: The Mission Approach. In: Higuera-Toledano, M.T., Wellings, A.J. (eds.) Distributed, Embedded and Real-time Java Systems, pp. 199–233. Springer US (2012)

[44] IBM: IBM WebSphere Real Time. `http://www-03.ibm.com/software/products/us/en/real-time` (2013), accessed: 2017-2-4

[45] IBM: RTSJ Reference Implementation (RI) and Technology Compatibility Kit (TCK). `http://www.timesys.com` (2013), accessed: 2017-2-4

[46] IceLab: IceLab. `http://icelab.dk/`, accessed: 2016-6-14

[47] ISO/IEC: Guidance for the use of the Ada Programming Language in High Integrity Systems. Tech. rep., ISO/IEC (2000)

[48] J Kwon: Ravenscar-Java: Java Technology for High Integrity Real-Time Systems. Ph.D. thesis, The University of York (2006)

[49] Jaffe, M.S., Busser, R., Daniels, D., Delseny, H., Romanski, G.: Progress Report on Some Proposed Upgrades to the Conceptual Underpinnings of DO-178B/ED-12B. In: System Safety, 2008 3rd IET International Conference on. pp. 1–6 (Oct 2008)

[50] James, P., Trumble, M., Treharne, H., Roggenbach, M., Schneider, S.: OnTrack: An Open Tooling Environment for Railway Verification. In: Brat, G., Rungta, N., Venet, A. (eds.) NASA Formal Methods, Lecture Notes in Computer Science, vol. 7871, pp. 435–440. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)

[51] Kalibera, T., Parizek, P., Malohlava, M., Schoeberl, M.: Exhaustive Testing of Safety-Critical Java. In: Proceedings of the 8th International Workshop on Java Technologies for Real-Time and Embedded Systems. pp. 164–174. JTRES '10, ACM, New York, NY, USA (19 Aug 2010)

[52] Kornecki, A., Zalewski, J.: Certification of Software for Real-Time Safety-Critical Systems: State of the Art. Innov. Syst. Softw. Eng. 5(2), 149–161 (2 Jun 2009)

[53] Leavens, G.T., Poll, E., Clifton, C., Cheon, Y., Ruby, C., Cok, D., Müller, P., Kiniry, J., Chalin, P., Zimmerman, D.M., Dietl, W.: JML Reference Manual (2003)

[54] Leuschel, M., Butler, M.: ProB: A Model Checker for B. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003: Formal Methods. pp. 855–874. Lecture Notes in Computer Science, Springer Berlin Heidelberg (8 Sep 2003)

[55] Lindner, A.: ANSI-C in Safety Critical Applications Lessons-Learned from Software Evaluation. In: Ehrenberger, W. (ed.) Computer Safety, Reliability and Security, Lecture Notes in Computer Science, vol. 1516, pp. 209–217. Springer Berlin Heidelberg (5 Oct 1998)

[56] Luckcuck, M., Wellings, A.J., Cavalcanti, A.: Safety-Critical Java: Level 2 in Practice. Concurr. Comput. (2016)

[57] Marriott, C.: Checking Memory Safety of Level 1 Safety-Critical Java Programs using Static-Analysis without Annotations. Ph.D. thesis, University of York (Sep 2014)

[58] Marriott, C., Cavalcanti, A.: SCJ: Memory-Safety Checking without Annotations. In: FM 2014: Formal Methods, pp. 465–480. Lecture Notes in Computer Science, Springer International Publishing (12 May 2014)

[59] Miyazawa, A., Cavalcanti, A.: SCJ-Circus: a Refinement-Oriented Formal Notation for Safety-Critical Java (7 Jun 2016)

[60] Motor Industry Software Reliability Association: MISRA C:2012: Guidelines for the Use of the C Language in Critical Systems. Motor Industry Research Association (2013)

[61] Oliveira, M., Cavalcanti, A.: From Circus to JCSP. In: Formal Methods and Software Engineering, pp. 320–340. Lecture Notes in Computer Science, Springer Berlin Heidelberg (8 Nov 2004)

[62] Oliveira, M., Cavalcanti, A., Woodcock, J.: A UTP Semantics for Circus. Form. Asp. Comput. 21(1-2), 3–32 (4 Dec 2007)

[63] Pizlo, F., Fox, J.M., Holmes, D., Vitek, J.: Real-Time Java Scoped Memory: Design Patterns and Semantics. In: Object-Oriented Real-Time Distributed Computing, 2004. Proceedings. Seventh IEEE International Symposium on. pp. 101–110 (May 2004)

[64] Plsek, A., Zhao, L., Sahin, V.H., Tang, D., Kalibera, T., Vitek, J.: Developing Safety Critical Java Applications with oSCJ/L0. In: Proceedings of the 8th International Workshop on Java Technologies for Real-Time and Embedded Systems. pp. 95–101. JTRES '10, ACM, New York, NY, USA (19 Aug 2010)

[65] Project, C.Z.T.: Community Z Tools. `http://czt.sourceforge.net/` (9 Apr 2016), accessed: 2016-9-25

[66] Real, J., Crespo, A.: Mode Change Protocols for Real-Time Systems: A Survey and a New Proposal. Real-Time Syst. 26(2), 161–197 (2004)

[67] Rios, J.R., Schoeberl, M.: Hardware Support for Safety-Critical Java Scope Checks. In: 2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing. pp. 31–38. ieeexplore.ieee.org (Apr 2012)

[68] RTCA: Formal Methods Supplement to DO-178C and DO-278A. Tech. rep. (2011)

[69] RTCA: Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A. Tech. rep. (2011)

[70] Sandén, B.I.: Real-Time Programming Safety in Java and Ada. Ada User J XXIII(2), 105–113 (2003)

[71] Schoeberl, M.: Real-Time Garbage Collection for Java. In: Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06). pp. 9 pp.– (Apr 2006)

[72] Schoeberl, M.: A Java Processor Architecture for Embedded Real-Time Systems. Int. J. High Perform. Syst. Archit. 54(1–2), 265–286 (Jan 2008)

[73] Siebert, F.: Hard Real-Time Garbage Collection in the Jamaica Virtual Machine. In: Real-Time Computing Systems and Applications, 1999. RTCSA '99. Sixth International Conference on. pp. 96–102 (1999)

[74] Singh, N.K., Wellings, A.J., Cavalcanti, A.: The Cardiac Pacemaker Case Study and its Implementation in Safety-critical Java and Ravenscar Ada. In: Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems. pp. 62–71. JTRES '12, ACM, New York, NY, USA (2012)

[75] Søndergaard, H., Korsholm, S.E., Ravn, A.P.: Safety-critical Java for Low-End Embedded Platforms. In: Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems. pp. 44–53. JTRES '12, ACM, New York, NY, USA (2012)

[76] Strøm, T.B., Schoeberl, M.: A Desktop 3D Printer in Safety-Critical Java. In: Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems. pp. 72–79. JTRES '12, ACM, New York, NY, USA (24 Oct 2012)

[77] Tang, D., Plsek, A., Vitek, J.: Static Checking of Safety Critical Java Annotations. In: Proceedings of the 8th International Workshop on Java Technologies for Real-Time and Embedded Systems. pp. 148–154. JTRES '10, ACM, Prague, Czech Republic (2010)

[78] The Open Group: Safety-Critical Java Technology Specification v0.94. Tech. Rep. v0.94, The Open Group (25 Jun 2013)

[79] The Open Group: Safety-Critical Java Technology Specification v0.100. Tech. Rep. v0.100, The Open Group (27 Dec 2014)

[80] Thomsen, B., Luckow, K.S., Leth, L., Bøgholm, T.: From Safety Critical Java Programs to Timed Process Models. In: Programming Languages with Applications to Biology and Security, pp. 319–338. Lecture Notes in Computer Science, Springer International Publishing (2015)

[81] Tindell, K.W., Burns, A., Wellings, A.J.: Mode Changes in Priority Preemptively Scheduled Systems. In: Real-Time Systems Symposium, 1992. pp. 100–109 (1992)

[82] Treharne, H., Schneider, S.: Using a Process Algebra to control B OPERATIONS. In: Araki, K., Galloway, A., Taguchi, K. (eds.) IFM'99, pp. 437–456. Springer London (1999)

[83] Treharne, H., Turner, E., Paige, R.F., Kolovos, D.S.: Automatic Generation of Integrated Formal Models Corresponding to UML System Models. In: Oriol, M., Meyer, B. (eds.) Objects, Components, Models and Patterns, Lecture Notes in Business Information Processing, vol. 33, pp. 357–367. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)

[84] U.S. Department of Defense: Department of Defense Requirements for High Order Computing Programming Langauges: Steelman (1978)

[85] Visser, W., Havelund, K., Brat, G., Park, S., Lerda, F.: Model Checking Programs. In: Automated Software Engineering Journal. vol. 10, pp. 3–12 (2003)

[86] Wei, K., Woodcock, J., Cavalcanti, A.: New Circus Time. University of York, Tech. Rep. , February (2012)

[87] Wellings, A.J., Kim, M.: Processing Group Parameters in the Real-Time Specification for Java. In: Proceedings of the 6th international workshop on Java technologies for real-time and embedded systems. pp. 3–9 (2008)

[88] Wellings, A.J.: Concurrent and Real-Time Programming in Java. John Wiley & Sons (2005)

[89] Wellings, A.J., Luckcuck, M., Cavalcanti, A.: Safety-Critical Java Level 2: Motivations, Example Applications and Issues. In: Proceedings of the 11th International Workshop on Java Technologies for Real-time and Embedded Systems. pp. 48–57. JTRES '13, ACM, New York, NY, USA (9 Oct 2013)

[90] Wheeler, D.A.: Ada, C, C++, and Java vs. the Steelman. ACM SIGAda Ada Letters XVII(4), 88–112 (1 Jul 1997)

[91] Ye, K., Woodcock, J.: Model Checking of State-Rich Formalism by Linking to CSP $\parallel$ B. Int. J. Softw. Tools Technol. Trans. (3 Nov 2015)

[92] Zeyda, F., Cavalcanti, A., Wellings, A.J., Woodcock, J., Wei, K.: Refinement of the Parallel CDx. Tech. rep., University of York (2012)

[93] Zeyda, F., Lalkhumsanga, L., Cavalcanti, A., Wellings, A.J.: Circus Models for Safety-Critical Java Programs. Comput. J. 57(7), 1046–1091 (1 Jul 2014)