TUCS

Sergey Ostroumov | Marina Waldén

Facilitating Formal Event-B Development by Visual Component-based Design

TURKU CENTRE *for* COMPUTER SCIENCE

TUCS Technical Report
No 1148, November 2015

Facilitating Formal Event-B Development by Visual Component-based Design

Sergey Ostroumov
TUCS – Turku Centre for Computer Science
Åbo Akademi University, Department of Information Technologies
Joukahaisenkatu 3-5A, 20520, Turku, Finland
Sergey.Ostroumov@abo.fi

Marina Waldén
Åbo Akademi University, Department of Information Technologies
Joukahaisenkatu 3-5A, 20520, Turku, Finland
Marina.Walden@abo.fi

TUCS Technical Report
No 1148, November 2015

Abstract

Due to the ever increasing complexity and criticality of modern systems, their correctness has to be evidently shown. This can be achieved by the use of formal methods such as Event-B. The development in Event-B follows the refinement approach, in which the specification is created top-down starting from a non-deterministic model and ending in a precise implementable one. The specification process is supported by theorem proving, so that one can guarantee correctness of the specification with respect to postulated properties called invariants. On the other hand, the formal modelling is limited in terms of reusability and bottom-up scalability. In addition, the formal Event-B specification of a system requires background knowledge, which prevents a fruitful communication between the developer and the customer.

This paper presents an approach that aims to facilitate scalability and reusability of formal development in Event-B as well as to enhance communication between the developer and the customer. The approach relies on the component-based design, where each component has a specific graphical representation. We present a set of the refinement patterns which support scalability and provide the connectivity (composition) between the components following the refinement approach. Our goal is to merge the top-down (refinement) and bottom-up (component-based development) approaches in order to improve rigorous Event-B specifications by visual representation. Eventually, the developers obtain the specification of a system that consists of two layers: logical and visual. The logical layer is fully based on the Event-B mathematical engine which gives the correctness proof. The visual layer is added on top of the logical layer, which gives a graphical representation of the Event-B specification.

Keywords: Event-B, Visual Design, Human-Machine Interface, Components Library, Formal Components, Refinement Patterns

TUCS Laboratory
RITES – Resilient IT Infrastructures
Distributed Systems Laboratory
Integrated Design of Quality Systems group

The work was done within the Advices project funded by Academy of Finland, grant No. 266373.

Contents

1. Introduction

Event-B [1] is a formal method that allows designers to build systems in such a manner that the correctness of the development process is supported by mathematical proofs. The development process proceeds in a top-down fashion starting from an abstract (usually non-deterministic) specification. This specification is then refined by stepwise unfolding the details about the system until the implementable level is reached. The process of transforming an abstract specification into an implementable one via a number of correctness preserving steps is known as refinement [2]. This mechanism allows the developers to build systems in a stepwise and correct-by-construction manner.

The specification (or the model) of a system in Event-B captures the functional behaviour as well as the essential properties that must hold (invariants). The refinement approach helps the designers to deal with the system requirements in a stepwise manner, which makes the correctness proof along the development easier. However, as more details are added to the system specification, it becomes complex and hard to manage, which limits the scalability of this approach. Moreover, the more details present in the specification, the harder it is to convince the stake holders about the fact that the system specification takes into account the necessary requirements and correctly specifies them.

To address these problems and facilitate easier system design and communication between the consumer and the developers, we propose an approach to component-based design within Event-B. This approach aims to combine top-down refinement and bottom-up component-based development approaches in order to provide a high level of scalability when designing complex systems in a rigorous manner. The approach relies on the formal library of parameterized visual components (see [22]), where components are added to the specification by the use of the “drag-and-drop” mechanism. We present a set of the refinement patterns that enable seamless integration of the components into a system. The development of the system is then reduced to manipulations with symbols whilst the correctness proof is supported by the underlying Event-B engine. The visual design eases the development effort, improves scalability and reusability as well as facilitates a fruitful communication between the developer and the customer.

The remainder of the paper is organized as follows. The next section describes the notation of Event-B and proof obligations that provide the correctness proof. Section 3 presents our approach to composition of the instantiated library components through refinement patterns. Section 4 illustrates the application of the proposed approach by the use of a case study from the avionics domain. Section 5 gives an overview on the existing visualisation approaches for the Event-B formalism. Finally, Section 6 concludes the paper the outlines the directions of the future work.

2. Preliminaries: Event-B

The Event-B formalism [1] offers several advantages. First, it allows us to build system level models. Second, it supports the refinement approach such that a model is built top-down in a correct-by-construction manner. Third, the development follows rigorous rules with mathematical proofs of correctness of models. Last but not least, it has a mature tool support extensible in the form of plug-ins, namely the Rodin platform [18]. Let us now describe the structure and notation of Event-B.

2.1. Event-B Model Structure

A specification in Event-B consists of *contexts* and *machines*. The relationship between them is shown in Figure 1. A context can be *extended* by another context whilst a machine can be *refined* by another machine. Moreover, a machine can refer to the contents of the context (to “*see*”).

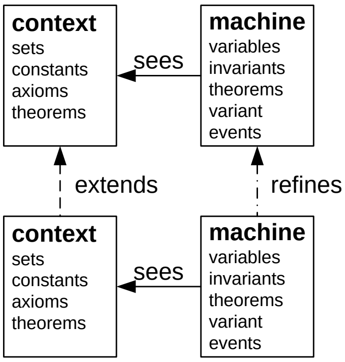

Figure 1. Event-B contexts and machines: contents and relationship [1]

A context specifies static structures such as data types in terms of *sets*, *constants*, properties given as a set of *axioms*. One can also postulate and prove *theorems* that ease proving effort during the model development.

A machine models the behaviour of a system. The machine includes *state variables*, *theorems*, *invariants*, a *variant* and guarded transitions (*events*). The invariants represent constraining predicates that define types of the state variables as well as essential properties of the system. The overall system invariant is defined as the conjunction of these predicates.

A variant is a natural number or a finite set. It is required to show the termination of certain events that can be executed several times in a row, e.g., modelling a loop.

An event describes a transition from a state to a state. The syntax of the event is as follows:

$$\mathsf{E} = \textbf{ANY}\ \mathsf{x}\ \textbf{WHERE}\ \mathsf{g}\ \textbf{THEN}\ \mathsf{a}\ \textbf{END}$$

where x is a list of event local variables. The *guard* g stands for a conjunction of predicates over the state variables and the local variables. The *action* a describes a collection of assignments to the state variables.

We can observe that an event models a guarded transition. When the guard g holds, the transition can take place. In case several guards hold simultaneously, any of the enabled transitions can be chosen for execution non-deterministically. If none of the guards holds, there is a deadlock.

When a transition takes place, the action a is performed. The action a is a composition of the assignments to the state variables executed simultaneously and denoted as $||$. An assignment can be either deterministic or non-deterministic. A deterministic assignment is defined as $v := E(w)$, where v is a list of state variables, E is a list of expressions over some set of state variables w. A non-deterministic assignment is specified as $v :| Q(w, v')$, where $Q(w, v')$ is a predicate over some state variables w and a new value v' of variable v. The variable v obtains such a value v' that $Q(w, v')$ holds.

2.2. Event-B Proof Mechanism

These denotations allow for describing semantics of Event-B in terms of *before-after predicates* (BA) [19]. Essentially, a transition is a BA that establishes a relationship between the model state before (v) and after (v') the execution of an event. Hence, the correctness of the model is verified by checking if the events preserve the invariants (INV) and are feasible to execute (FIS) in case the event action is non-deterministic:

$$Inv \wedge g_e \Rightarrow [BA_e]Inv \qquad \text{(INV)}$$

$$Inv \wedge g_e \Rightarrow \exists v' \,.\, BA_e \qquad \text{(FIS)}$$

where Inv is a model invariant, g_e and BA_e are the guard and the before-after predicate of the event e, respectively. The expression $[BA_e]Inv$ stands for the substitution in the invariant Inv according to BA_e.

In addition, deadlock freedom of the specification may be corroborated. A deadlock free specification stands for the case where there exists at least one event that can be executed. To achieve this, one needs to postulate a machine theorem that includes the guards of all the events connected with disjunction and show that the proof obligation (DLF) [1] is preserved:

$$\forall S, C, V \,.\, A \wedge I \Rightarrow \bigvee_{i=1}^{n} g_i \qquad \text{(DLF)}$$

where n is the number of events and g_i is the guard of the i-th event. The structures S, C and A represent sets, a collection of constants and axioms introduced into a context, respectively. The structures V and I stand for a set of state variables and a set of invariants of a machine, respectively.

2.3. Refinement in Event-B

Since the specification development in Event-B follows the refinement approach, one has to prove that the more concrete (refined) events simulate their abstract counterparts. To show this, the refined events must preserve the guard strengthening (GRD) and action simulation (SIM) proof obligations [20] as well:

$$\forall S, C, S_r, C_r, V, V_r, x, x_r \,.\, A \wedge A_r \wedge I \wedge I_r \wedge g_r \Rightarrow g \qquad \text{(GRD)}$$

$$\forall S, C, S_r, C_r, V, V_r, x, x_r \,.\, A \wedge A_r \wedge I \wedge I_r \wedge BA_{er} \Rightarrow BA_e \qquad \text{(SIM)}$$

where all letters with subscript “r” stand for the refined versions of the aforementioned structures.

To prove that new events executed several times in a row terminate, one also has to show that these events are consistent with a variant. In particular, these events have to preserve either of the following proof obligations depending on whether the variant is a natural number (VAR_N) or a finite set (VAR_S) [20]:

$$\forall S, C, V \,.\, A \wedge I \Rightarrow Var \in \mathbb{N} \wedge [BA_e]Var < Var \qquad \text{(VAR\_N)}$$

$$\forall S, C, V \,.\, A \wedge I \Rightarrow finite(Var) \wedge card([BA_e]Var) < card(Var) \qquad \text{(VAR\_S)}$$

where Var is a variant that denotes a numeric expression or a finite set of values. The expressions $finite(Var)$ and $card(Var)$ specify finitness and cardinality of the set variant, respectively.

In case the model needs to be deadlock free, one can show the relative deadlock freedom, i.e., all concrete events should not deadlock more frequently than the abstract ones. Therefore, the disjunction of the abstract guards should imply the disjunction of the concrete guards (proof obligation (DLFR)) [1]:

$$\forall S, C, V \,.\, A \wedge I \wedge I_r \wedge \bigvee_{i=1}^{n} g_i \Rightarrow \bigvee_{j=1}^{m} g_{rj} \qquad \text{(DLFR)}$$

where m is the number of concrete events and g_j is the guard of the j-th event.

The Rodin platform [18], a tool support for Event-B, automatically generates and attempts to discharge (prove) the necessary proof obligations. The best practices encompass the development of the specification in such a manner that 90-95% of the proof obligations are discharged automatically. However, the tool sometimes requires the user assistance provided via the interactive prover. Typically, the claims that are difficult for the automatic prover to discharge require case distinction and/or data substitution.

3. Composition of Components

We rely on the formal library of visual components described in [22]. Whenever needed, a designer picks and instantiates the necessary components to form a system. However, these components have to be connected in order to fulfil the requirements and perform the mission. We now present the connection patterns that enable the composition of the instantiated components into a system through refinement.

The overall components composition approach is shown in Figure 2. The idea behind this composition is that the model of the system consolidates the interfaces of the necessary components and the connections between them. The functional events that comprise the bodies of the components are left in separate machines included into the system specification. This mechanism provides the structure of the system model.

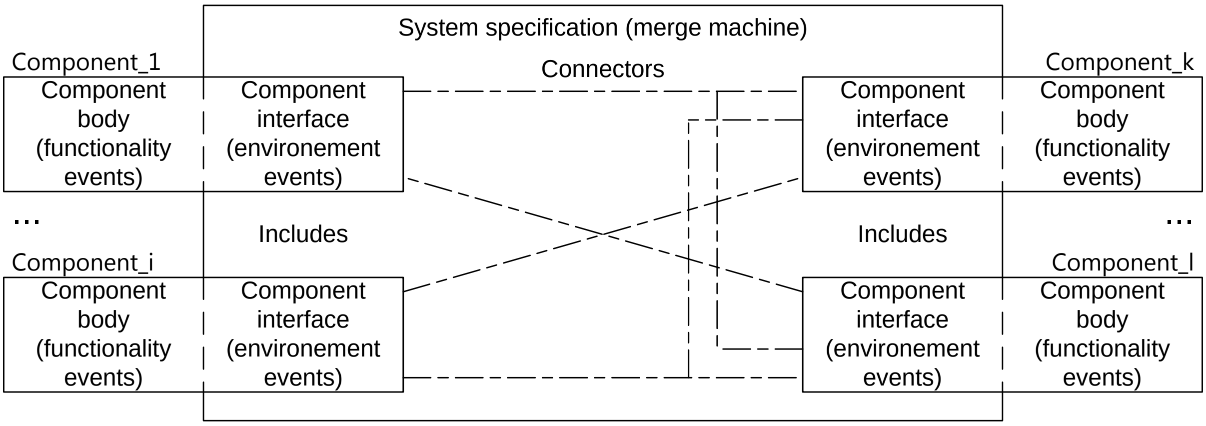

Figure 2. Overall composition diagram

The components are connected using connectors (Figure 3, a)). A simple connector is a variable that is updated by one component and is read by another one. Once the source component (Component_i in Figure 3, a)) has produced a new value and this value has been promoted to the connector, this component can read the new input and produce the new output. Once the value on the connector is updated, the destination component (Component_k in Figure 3, a)) can read it and produce the output. Therefore, the source and the destination components can work in parallel, even though they are connected sequentially (i.e., the source component affects the output of the destination component) (Figure 3, b)).

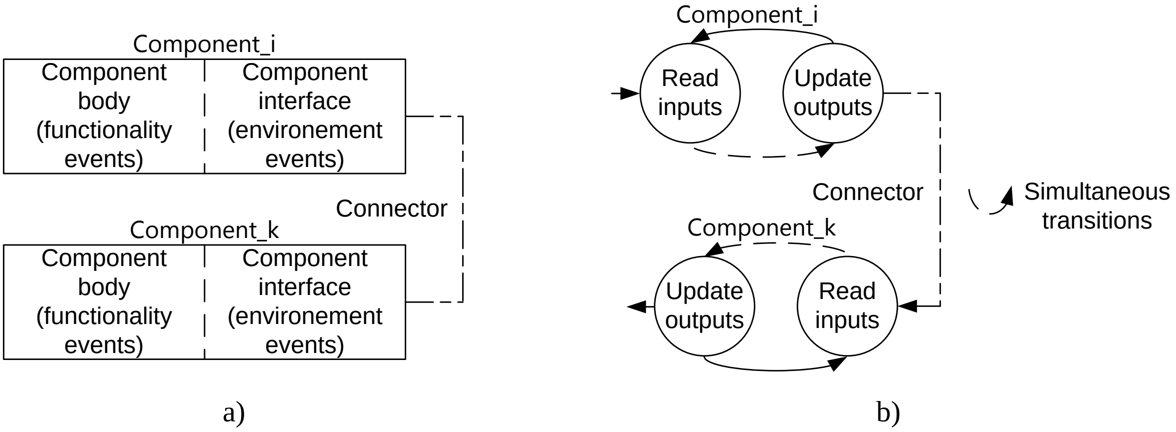

Figure 3. Sequential connection of the components: a) structure, b) automata

3.1. Refinement Pattern for Introducing a Connector

The connection between the components is performed in a stepwise manner by refinement. This eases the proof effort and allows us to ensure the correct behaviour of the composed machine. The first step in connecting the components is to introduce a connector after the source component is specified (Figure 4). As described above, the source and the destination components are connected sequentially, such that the update of the output of the source component affects the input of the destination component.

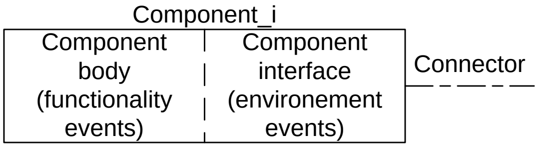

Figure 4. A source component with a connector

The complete pattern for introducing a connector can be found in Appendix A. From now on, we will use the following convention in naming Event-B elements and labels: name_<i_>S_n_<j>, where name is the name of an element, i_ and j are the optional numerations and S ∊ {C,M,r}, C – context, M – machine, r or R – refinement, n is the number of a refinement step.

To introduce this behaviour, we start by defining a set which we call the control set (Listing 1). This set provides the mechanism to advance between the source component, connector and the destination component as shown below.

Listing 1. The control set for the connector pattern

```
context System_Connection_C_{n-1} extends Component_i_C_{n-2}
constants
  SYSTEM_CONTROL_R_{n-1}
axioms
  SYSTEM_CONTROL_R_{n-1} = {0,1,2}
end
```

In the simplest case, the connector can be modelled with a single variable. Hence, the composition machine embodies the variables derived from the source component and at least two new ones. These are the control (system_control_r$_{n-1}$) and the connector (system_connection_Component_i_Component_k_r$_{n-1}$) as shown in Listing 2. Additionally, the composition machine has a variant to prove the convergence of the event that models the value update on the connector variable. The convergent event decreases the value of the control variable from 1 to 0, i.e., the variant is decreased.

Listing 2. The connector and control variables

```
machine System_Connection_M_{n-1} refines Component_i_M_{n-2} sees System_Connection_C_{n-1}
variables …
  system_control_r_{n-1}
  system_connection_Component_i_Component_k_r_{n-1}
invariants
  system_control_r_{n-1} ∊ SYSTEM_CONTROL_R_{n-1} ∧
  system_connection_Component_i_Component_k_r_{n-1} ∊ <COMPONENT_i_OUTPUT_TYPE>
variant system_control_r_{n-1}
```

The initial values of the control and the connector variables are zero and some initial value derived from the source component, respectively. Listing 3 summarizes the initialization of these variables.

Listing 3. Initialization of the connector and control variables

```
  event INITIALISATION extends INITIALISATION
    then
      system_control_r_{n-1} ≔ 0 || system_connection_Component_i_Component_k_r_{n-1} ≔ <INIT_VALUE>
  end
```

The environmental event of the source component is refined considering the aforementioned control flow (Listing 4). That is, the component can read the new input when its current output value has been promoted to the connector (system_control_r$_{n-1}$ = 0).

Listing 4. Refinement of the environment event of the component i

```
  event Component_i_environment refines Component_i_environment
    where
      … ∧ // Other guards derived from the component i
      system_control_r_{n-1} = 0
    then
      … || // Other actions derived from the component i
```

```
   system_control_r_{n-1} ≔ 1
 end
```

The value of the connector is updated when the source component has produced the output (<Component_i_mode> = 0 in Listing 5). The mode of the component can also be of the Boolean type, if there are two alternating modes, in which case 0 ⇔ FALSE, 1 ⇔ TRUE. Notice that this event is convergent with, since it must terminate and return the control to the source component. The convergence is proved on the basis of the control variable (system_control_r$_{n-1}$) whose value is changed from 1 to 0, i.e., is decreased.

Listing 5. The connector event

```
 convergent event system_connection_Component_i_Component_k
  where
   system_control_r_{n-1} = 1 ∧
    // We need to be sure that the component i has updated its outputs
   <Component_i_mode> = 0 // the component mode can also be of Boolean type (0 ⇔ FALSE, 1 ⇔ TRUE)
  then
   system_control_r_{n-1} ≔ 0 || system_connection_Component_i_Component_k_r_{n-1} ≔ <Component_i_Output>
 end
```

3.2. Refinement Pattern for Introducing a Destination (Generic) Component

After introducing the connector, the destination component can be added to the system using the refinement approach, so that we can obtain the model as shown in Figure 3. We will illustrate this by an example of the addition of the generic component. As in the previous pattern, we start with the introduction of the control set into the context (Listing 6). Since a component can have parameters, they are also instantiated and introduced into this context as will be shown in the case study section. The complete pattern can be found in Appendix B.

Listing 6. The control set for the component k pattern

```
context Component_k_Parameters_C_n extends System_Connection_C_{n-1}
constants
 SYSTEM_CONTROL_R_n
axioms
 … // Parameters of component k, if any
 SYSTEM_CONTROL_R_n = {0,1,2}
end
```

From now on, we omit the data related to the generic component per se and only focus on the parts that change according to the proposed pattern. Similarly to the previous pattern, the pattern for introducing a destination (generic) component also has a variant. Due to the refinement relation, the control variable cannot be modified. Thus, it has to be replaced with a new control variable, namely system_control_r$_n$ (Listing 7), which simulates the old control variable according to the gluing invariants system_control_r$_{n-1}$ = 1 ⇔ system_control_r$_n$ = 1 and system_control_r$_{n-1}$ = 0 ⇔ system_control_r$_n$ = 0 ∨ system_control_r$_n$ = 2.

Listing 7. The variables and the properties of the component introduction pattern

```
machine Component_k_M_n refines System_Connection_M_{n-1} sees Component_k_Parameters_C_n
variables …
 system_connection_Component_i_Component_k_r_{n-1}
~~ system_control_r_{n-1}~~
 GenericComponent_k_I
 GenericComponent_k_O
 GenericComponent_k_mode
 GenericComponent_k_IOrelation
 system_control_r_n
```

invariants
… ∧ *// The types and the properties of the generic component*
system_control_r$_{n}$ ∈ SYSTEM_CONTROL_R$_{n}$ ∧ (system_control_r$_{n-1}$ = 1 ⇔ system_control_r$_{n}$ = 1) ∧
(system_control_r$_{n-1}$ = 0 ⇔ system_control_r$_{n}$ = 0 ∨ system_control_r$_{n}$ = 2)
variant system_control_r$_{n}$

At the beginning, all the variables receive the aforementioned initial values. The environment event of the component i and the event modelling the connection between component i and component k are refined by simply replacing the old control variable with the new one as shown in Listing 8 and Listing 9, respectively. The other guards and actions remain unchanged.

Listing 8. Refinement of the component i environment event

event Component_i_environment refines Component_i_environment
where
… *// Other guards derived from the component i*
~~system_control_r$_{n-1}$ = 0~~ ∧ system_control_r$_{n}$ = 0
then
… *// Other actions derived from the component i*
~~system_control_r$_{n-1}$ ≔ 1~~ || system_control_r$_{n}$ ≔ 1
end

Listing 9. Refinement of the connection event

event system_connection_Component_i_Component_k refines system_connection_Component_i_Component_k
where
<Component_i_mode> = 0 ∧ system_control_r$_{n}$ = 1
then
system_connection_Component_i_Component_k_r$_{n-1}$ ≔ <Component_i_Output_Value> || system_control_r$_{n}$ ≔ 2
end

The guard of the environment event of the destination generic component (Listing 10) that is being introduced is strengthened by checking the control variable if the component can read the input (system_control_r$_{n}$ = 2). Once the component reads the input, it returns the control back, so that the new iteration of reading the input and updating the output can take place (system_control_r$_{n}$ ≔ 0). Notice that this is event is convergent, i.e., it must terminate and return the control to the source component.

Listing 10. Introduction of the environment event the generic component k

convergent event GenericComponent_k_environment
where
GenericComponent_k_mode = 0 ∧ system_control_r$_{n}$ = 2
then
GenericComponent_k_mode ≔ 1 || GenericComponent_k_I ≔ <SET_OF_OUTPUT_VALUES_OF_COMPONENT_i> ||
system_control_r$_{n}$ ≔ 0
end

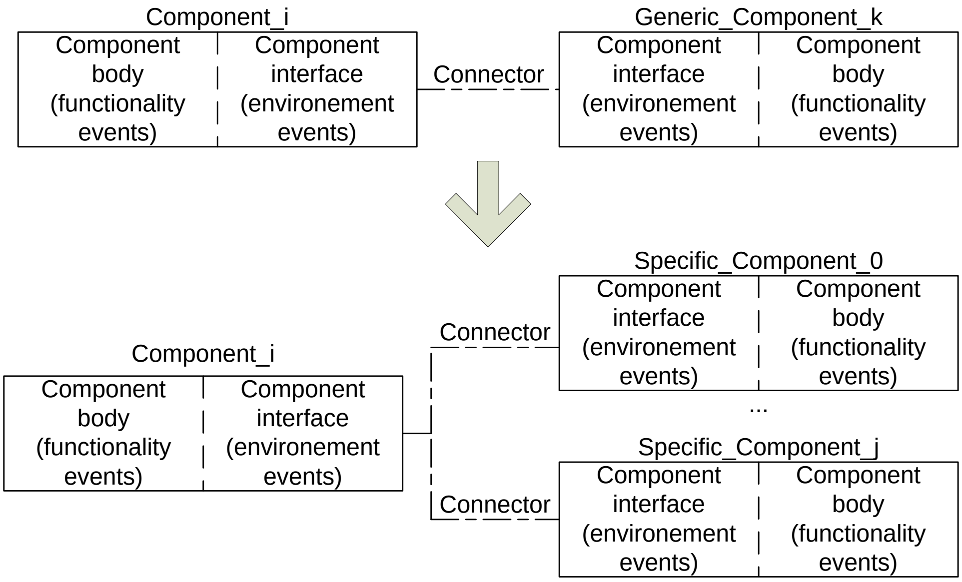

Figure 5. Refinement of a generic component into a set of specific components

3.3. Refinement Pattern: Generic Component into a Set of Specific Ones

Once the design decisions are made and the components that need to be placed instead of the generic one are known, the generic component can be refined into a set of specific ones. We assume that the specific components work independently of each other (i.e., there are no connection between them), but they are sequentially connected to the source component i (Figure 5). The complete pattern can be found in Appendix C. Notice that the same pattern can be applied in case when different components that refine the generic one are connected to different source components.

The first step when refining the generic component into a collection of specific ones is to specify a set of the specific components that are to replace the generic one. We define the set of the specific components ($COMPONENTS\_R_{n+1}$) by using the keyword partition (Listing 11). This context also embodies the instantiated parameters of the specific components.

Listing 11. Introduction of the set of specific components

```
context Specific_Components_Parameters_C_{n+1} extends Component_k_Parameters_C_{n}
sets
  COMPONENTS_R_{n+1}
constants ...
  component_0_r_{n+1} ...
  component_j_r_{n+1}
axioms
  ... // Other parameters of the components
  partition(COMPONENTS_R_{n+1}, {component_0_r_{n+1} }, ..., {component_j_r_{n+1}})
end
```

Due to the fact that each specific component being introduced has a definite set of inputs and outputs as well as a relation between them, they replace the generic input, output and relation, respectively (Listing 12). To allow a component to read the input once per each iteration, we use a special variable, namely $components\_read\_r_{n+1}$, which maps a component label to 0 or 1. The value 0 means that the component has not read the input yet whilst the value 1 stands the opposite case. Moreover, each component requires its own connector to read the input from the source component. Thus, these variables are data refined.

Listing 12. State variables of the refined machine

machine Specific_Components_M$_{n}$ refines M$_{n-1}$_Component_k sees Specific_Components_Parameters_C$_{n}$
variables …
~~GenericComponent_k_I~~
~~GenericComponent_k_O~~
~~GenericComponent_k_IOrelation~~
~~system_connection_Component_i_Component_k_r$_{n-1}$~~
GenericComponent_k_mode
system_connection_i_0_r$_{n+1}$
… *// Similar connectors to other components*
system _connection_i_j_r$_{n+1}$
components_read_r$_{n+1}$
invariants
… *// The properties of the specific components*
components_read_r$_{n+1}$ ∈ COMPONENTS_R$_{n+1}$ → 0..1 ∧
system_connection_i_0_r$_{n+1}$ ∈ **<COMPONENT_0_INPUT_TYPE>** ∧
system_connection_i_j_r$_{n+1}$ ∈ **<COMPONENT_j_INPUT_TYPE>**

To simplify the refinement, we assume that all the input values of the generic component are a union of the input values of the specific components. The output value of the generic component is a union of the output values of the specific component. Finally, the generic relation is simply a Cartesian product of these two sets (Listing 13). The mode of the generic component after reading the input is equivalent to the case when all the specific components have read their inputs (system_control_r$_{n}$ = 0 ∧ GenericComponent_k_mode = 1 ⇒ components_read_r$_{n+1}$[COMPONENTS_R$_{n+1}$] = {0}). The generic connector is also split into a collection of specific connectors for each specific component (e.g., system_connection_Component_i_Component_k_r$_{n-1}$ = system_connection_i_0_r$_{n+1}$). Clearly, the environment events of the newly introduced components have to terminate, so that the source component can read the new input and update its outputs and, consequently, the value present on the connector. To support this, we provide a variant as shown in Listing 13.

Listing 13. Properties of the refinement of the generic component

GenericComponent_k_I ⊆ $\bigcup^{m}_{i=0}$INPUT_TYPE$_{i}$ ∧ GenericComponent_k_O ⊆ $\bigcup^{m}_{i=0}$OUTPUT_TYPE$_{i}$ ∧
GenericComponent_k_IOrelation ⊆ $\bigcup^{m}_{i=0}$INPUT_TYPE$_{i}$ × $\bigcup^{m}_{i=0}$OUTPUT_TYPE$_{i}$ ∧
(system_control_r$_{n}$ = 0 ∧ GenericComponent_k_mode = 1 ⇒ components_read_r$_{n+1}$[COMPONENTS_R$_{n+1}$] = {0}) ∧
system_connection_Component_i_Component_k_r$_{n-1}$ = system_connection_i_0_r$_{n+1}$ ∧
system_connection_Component_i_Component_k_r$_{n-1}$ = system_connection_i_j_r$_{n+1}$ ∧
variant card(COMPONENTS_R$_{n+1}$) − components_read_r$_{n+1}$(**component_0_r$_{n+1}$**) − … −
components_read_r$_{n+1}$(**component_j_r$_{n+1}$**)

The newly introduced data structures affect the initialisation (Listing 14). Particularly, we need to provide witnesses for the disappearing generic input, output and relation. The other variables are initialised as usual.

Listing 14. Initialisation: witnesses for the disappearing variables

event INITIALISATION
with
GenericComponent_k_I' = $\bigcup^{m}_{i=0}$INPUT_TYPE$_{i}$ ∧ GenericComponent_k_O' = $\bigcup^{m}_{i=0}$OUTPUT_TYPE$_{i}$ ∧
GenericComponent_k_IOrelation' = $\bigcup^{m}_{i=0}$INPUT_TYPE$_{i}$ × $\bigcup^{m}_{i=0}$OUTPUT_TYPE$_{i}$
then
… *// Initialization of other state variables from previous refinements*
system_connection_i_0_r$_{n+1}$ ≔ **<INIT_VALUE>** || system_connection_i_j_r$_{n+1}$ ≔ **<INIT_VALUE>** ||
components_read_r$_{n+1}$ ≔ COMPONENTS_R$_{n+1}$ × {0}
end

The environment event of component i is not affected by this refinement and, therefore, remains unchanged. On the other hand, the event modelling the update of the connector is refined considering the fact that there is a collection of connectors for each component (Listing 15).

Listing 15. Refinement of the connection event

```
event system_connection_i_0j refines system_connection_Component_i_Component_k
 where
  <Component_i_mode> = 0 ∧ system_control_r_n = 1
 then
  system_control_r_n ≔ 2 || system_connection_i_0_r_{n+1} ≔ <Component_i_Output> ||
  system_connection_i_j_r_{n+1} ≔ <Component_i_Output>
end
```

Next, we introduce all the environment events of the instantiated components as every component has to update its inputs according to the output of the source component. The environment events of the newly introduced specific components are augmented with the guards that take into account the control mechanism and restrict the number of inputs read to 1 per iteration (`components_read_r_{n+1}(component_0_r_{n+1}) = 0` in Listing 16 and `components_read_r_{n+1}(component_j_r_{n+1} ) = 0` in Listing 17). The inputs of these components receive the value from the corresponding connectors (e.g., `component_0_I_0 ≔ system_connection_i_0_r_{n+1}`). If a component has other inputs, they can also be updated here.

Listing 16. Environment event of the specific component 0

```
convergent event Component_0_environment
 where
  component_0_mode = 0 ∧ system_control_r_n = 2 ∧ components_read_r_{n+1}(component_0_r_{n+1} ) = 0
 then
  component_0_mode ≔1 || components_read_r_{n+1}(component_0_r_{n+1} ) ≔ 1 ||
  component_0_I_0 ≔ system_connection_i_0_r_{n+1} || … // Update of the other inputs not connected to component i
end
```

Listing 17. Environment event of the specific component j

```
convergent event Component_j_environment
 where
  component_j_mode = 0 ∧ system_control_r_n = 2 ∧ components_read_r_{n+1}(component_j_r_{n+1} ) = 0
 then
  component_j_mode ≔ 1 || components_read_r_{n+1}(component_j_r_{n+1} ) ≔ 1 ||
  component_j_I_0 ≔ system_connection_i_j_r_{n+1} ||  … // Update of the other inputs not connected to component i
end
```

Once all the components have read the input, all the control variables are reset as shown in Listing 18. When the control variables are set to the initial values, a new iteration can start.

Listing 18. Reset of the control variables to allow a new iteration

```
event GenericComponent_k_environment refines GenericComponent_k_environment
 where
  GenericComponent_k_mode = 0 ∧ system_control_r_n = 2 ∧ components_read_r_{n+1}[COMPONENTS_R_{n+1}] = {1}
 then
  GenericComponent_k_mode ≔ 1 || system_control_r_n ≔ 0 || components_read_r_{n+1} ≔ COMPONENTS_R_{n+1} × {0}
end
```

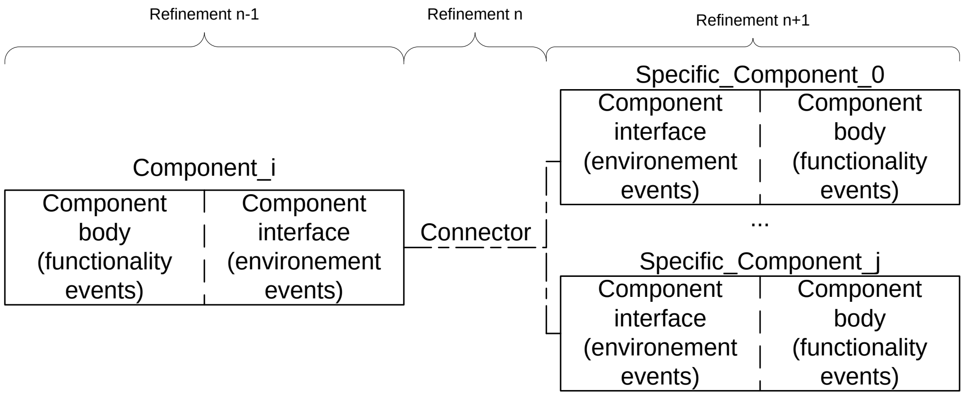

Figure 6. Introduction of specific components without adding the generic one

3.4. Refinement Pattern for the Introduction of Several Parallel Components without Introducing the Generic One

In some cases, when the developer knows which components need to be introduced at a refinement step, the developer can avoid the refinement with the generic component. Instead, one can refine the specification of the system, so that the necessary components are introduced directly (Figure 6). In this case, we propose the following pattern whose complete model can be found in Appendix D. Similarly to the pattern presented above, we start by specifying the set of the components to be introduced in the refinement (Listing 19).

Listing 19. Introduction of the components list

```
context Specific_components_Cn extends System_connection_Cn-1
sets
  COMPONENTS_Rn
constants
  SYSTEM_CONTROL_Rn
  component_0_rn
  component_j_rn
  … // Parameters of the components and component_q_rn, where q ∊ {0,…,j}
axioms
  SYSTEM_CONTROL_Rn = {0,1,2} ∧ partition(COMPONENTS_Rn, {component_0_rn}, …, {component_j_rn})
  … // Definitions of the components
end
```

Similarly as in the pattern described in Section 3.3, we use a special variable (components_read_r$_n$) to restrict reading of the inputs by the components once per iteration. Additionally, each component needs its own connector to read the input from, which refines the generic connector introduced earlier. Listing 20 summarizes the state variables and their data types.

Listing 20. Refinement state variables and their data types

```
machine Specific_Components_Mn refines System_Connection_Mn-1 sees Specific_components_Cn
variables
  system_control_rn
  components_read_rn
  system_connection_i_0_rn
  system_connection_i_j_rn
  … // State variables of the components and other connectors
invariants
  … // Properties of the components
  components_read_rn ∊ COMPONENTS_Rn → 0..1 ∧ system_control_rn ∊ SYSTEM_CONTROL_Rn ∧
  system_connection_i_0_rn ∊ <COMPONENT_0_INPUT_TYPE> ∧ /* system_connection_i_q_rn, where q ∊ {0,…,j} */ ∧
  system_connection_i_j_rn ∊ <COMPONENT_j_INPUT_TYPE>
```

The main properties of this refinement include the properties of the components being added to the specification as well as the gluing invariants shown in Listing 21. This refinement also contains a variant to show the termination of the environment events of the components being introduced.

Listing 21. Properties of the refinement without the generic component

```
 system_connection_i_k_r_{n-1} = system_connection_i_0_r_n ∧ system_connection_i_k_r_{n-1} = system_connection_i_j_r_n ∧
 (system_control_r_{n-1} = 1 ⇔ system_control_r_n = 1) ∧
 (system_control_r_{n-1} = 0 ⇔ system_control_r_n = 0 ∨ system_control_r_n = 2)
variant card(COMPONENTS_R_n) − components_read_r_n(component_0_r_n) − … − components_read_r_n(component_j_r_n)
```

Initially, the control variables are set to 0 in order to allow the execution from the source component. The connector variables are assigned some initial values according to their data types (Listing 22).

Listing 22. Initialisation of the control and connector variables

```
 event INITIALISATION
   then
     …
     system_control_r_n ≔ 0 || components_read_r_n ≔ COMPONENTS_R_n × {0} ||
     system_connection_i_0_r_n ≔ <INIT_VALUE> || system_connection_i_j_r_n ≔ <INIT_VALUE>
 end
```

The environment event of the source component i is refined considering the newly introduced control variables. Listing 23 illustrates the guards and actions after this refinement.

Listing 23. Refinement of the environment event of the source component i

```
 event Component_i_environment refines Component_i_environment
   where
     <Component_i_mode> = 0 ∧ system_control_r_n = 0
   then
     <Component_i_mode> ≔ 1 || system_control_r_n ≔ 1
     … // Read the new input
 end
```

Similarly as in the pattern described in Section 3.3, the generic connector is refined, so that each component is connected to its own connector. Listing 24 illustrates the refinement of the connection event. Notice that the special variable $\mathrm{components\_read\_r}_n$ is also reset in this event to allow the components to read the input at a new iteration.

Listing 24. Properties of the refinement without the generic component

```
 event system_connection_i_0…j refines system_connection_Component_i_Component_k
   where
     <Component_i_mode> = 0 ∧ system_control_r_n = 1
   then
     system_control_r_n ≔ 2 || components_read_r_n ≔ COMPONENTS_R_n × {0} ||
     system_connection_i_0_r_n ≔ <Component_i_Output> || system_connection_i_j_r_n ≔ <Component_i_Output>
 end
```

Next, we introduce all the environment events of the instantiated components as every component has to update its inputs according to the output of the source component. Listing 25 and Listing 26 show examples of the environment events of the components 0 and j, respectively. Notice that there can be several environment events, i.e., component_q_environment, where $q \in 0..j$. Since the components work in parallel and they do not refine the generic component, we need guards to limit their execution and properly establish the control flow. The guard $\mathrm{components\_read\_r}_n(\mathrm{component\_0\_r}_n) = 0$ in Listing 25 enable the component to read the inputs once per each iteration. Next, we determine if all the components except for the current one have read the inputs (components_read_r_n[COMPONENTS_R_n ∖ {**component_0_r_n**}] = {1}). In this case the control can be returned to the source component (⇒ *system_control_r_n_new* = 0). Otherwise, the control remains unchanged (¬components_read_r_n[COMPONENTS_R_n ∖ {**component_0_r_n**}] = {1} ⇒ *system_control_r_n_new* = 2).

Listing 25. Environment event of the component 0

```
convergent event component_0_environment
  any system_control_r_n_new
  where
   <Component_0_mode> = 0 ∧ system_control_r_n = 2 ∧ components_read_r_n(component_0_r_n) = 0 ∧
   (components_read_r_n[COMPONENTS_R_n∖{component_0_r_n}] = {1} ⇒ system_control_r_n_new = 0) ∧
   (¬components_read_r_n[COMPONENTS_R_n∖{component_0_r_n}] = {1} ⇒ system_control_r_n_new = 2)
  then
   <Component_0_mode> ≔ 1 || <Component_0_input> ≔ system_connection_i_0_r_n ||
   system_control_r_n ≔ system_control_r_n_new || components_read_r_n(component_0_r_n) ≔ 1 ||
   … /* Update the other inputs of the component 0, if any */
end
```

Listing 26. Environment event of the component j

```
convergent event component_j_environment
  any system_control_r_n_new
  where
   <Component_j_mode> = 0 ∧ system_control_r_n = 2 ∧ components_read_r_n(component_j_r_n) = 0 ∧
   (components_read_r_n[COMPONENTS_R_n∖{component_j_r_n}] = {1} ⇒ system_control_r_n_new = 0) ∧
   (¬components_read_r_n[COMPONENTS_R_n∖{component_j_r_n}] = {1} ⇒ system_control_r_n_new = 2)
  then
   <Component_j_mode> ≔ 1 || <Component_j_input> ≔ system_connection_i_j_r_n ||
   system_control_r_n ≔ system_control_r_n_new || components_read_r_n(component_j_r_n) ≔ 1 ||
   … // Update the other inputs of the component j, if any
end
```

3.5. Summary

The proposed patterns facilitate seamless integration between the components in a systematic fashion and ease the proving effort. The patterns allow the designers to introduce and connect components in a natural way and follow the refinement approach, so that the correctness of the system specification can be shown using the usual POs. In addition, the patterns 3.2 with 3.3 vs. 3.4 illustrate different approaches of the component-based system modelling depending on the design decisions made during the system development. The system development is then a combination of top-down (refinement) and bottom-up (components) approaches, where the developer manipulates (introduces) the components in the “drag-and-drop” fashion by manipulating visual symbols instead of text (see [22]). Notice that the visual layer is on top of the formal layer and it does not restrict a developer to add more properties to the model, if needed.

4. Case study

Let us demonstrate the proposed approach using the landing gear case study whose detailed description can be found in the proceedings of the ABZ workshop [21]. The system consists of a digital controller and a few actuators. The function of the system is to operate the landing gears and associated doors. Depending on the reactions from the pilot, the digital controller manipulates the mechanical part. The mechanical part, in its turn, consists of front, left and right landing sets. Each set includes a door, a landing gear and hydraulic cylinders that are attached to and move the corresponding doors and gears.

The architecture of the system is shown in Figure 7. The general electro-valve provides hydraulic power to the specific electro-valves from the aircraft hydraulic system. There are 4 specific electro-valves which set the pressure to the cylinders opening/closing the doors as well as to the cylinders extending/retracting the gears. Clearly, the position of the piston of a cylinder coincides with the position of the corresponding controlling component. For instance, if the front door cylinder is extended, the front door is open.

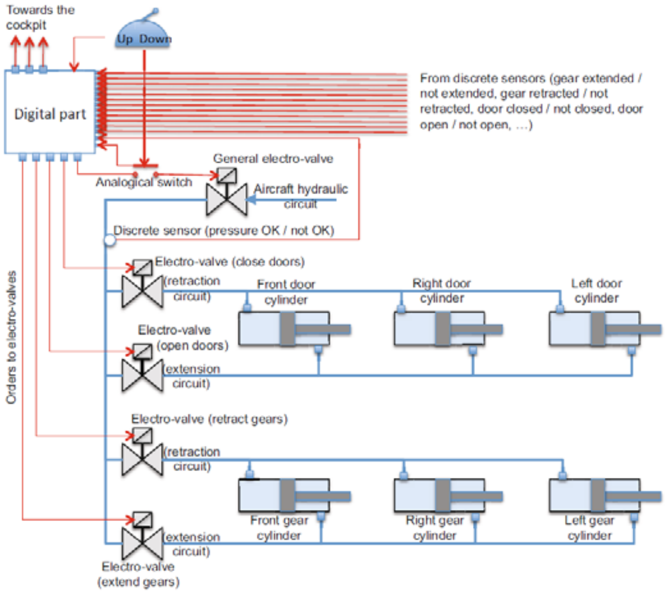

Figure 7. Architecture of the landing gear system [21]

We develop the part that consists of the general electro-valve, the specific electro-valves and the cylinders that manipulate the doors and gears. We start by introducing the general electro-valve which serves as the source component and proceed from top-left to bottom-right. Although the architecture of the system is known a priori, we will show the use of the generic component as if some parts were unknown during the refinement.

We first number the electro-valves for doors and gears from 0 to 3 from top to bottom, respectively, and cylinders from 0 to 5 from left-top to right-bottom, respectively. The refinement strategy for the development of the system is as follows. We will first instantiate the formal library component, namely the valve (see Section 4.2 in [22]), into the general electro-valve. Then, we will use the pattern described in Section 3.1 to introduce a connector between the general electro-valve and the specific electro-valves. However, instead of instantiating these valves directly, we will add the generic component to show the application of the pattern from Section 3.2 and of the model presented in Section 4.6 in [22]. Then, we will refine the derived model by replacing the generic component with specific electro-valves according to the pattern described in Section 3.3. Finally, we will illustrate the application of the refinement pattern without the generic component (Section 3.4) by adding a set of cylinders to the system specification.

4.1. Abstract specification: instantiation of the general electro-valve

As mentioned above, we start the development of the landing gear from the introduction of the general electro-valve. Consequently, pick the library component electro-valve (see [22]) and instantiate it into the general electro-valve by "dragging-and-dropping" its visual symbol (Figure 8) into the system specification. The instantiation stands for providing the precise values for the parameters and specifying the name. The name of the component is augmented with a sequence number (e.g., `GEV_0`, where `GEV` is the name and `_0` is the sequence number) as the specification can contain several components of the same kind. This number helps to distinguish between these components.

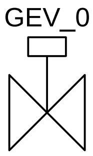

Figure 8. Abstract specification of the landing gear system

The partial description of the formal specification behind the visual representation is given below. The complete instantiated model including the environment and functional events for the illustration purpose can be found in Appendix E.

The values of the parameters that the developer has to provide depend on the component. For instance, Listing 27 illustrates the instantiation of the parameters of the general electro-valve. Particularly, we provide the values for the maximum diameter of the valve (GEV_0_diameter_max_val) which essentially defines the power of the flow and for the rate with which the valve opens and closes (GEV_0_rate). Some axioms of a parameterized component can become theorems (e.g., theorem GEV_0_rate ≤ GEV_0_diameter_max_val − GEV_0_diameter_min_val which is the axiom evalve_rate ≤ evalve_diameter_max_val − evalve_diameter_min_val from Section 4.2 in [22]) in order to support the welldefinedness of the instantiation.

Listing 27. General electro-valve instantiated parameters

```
context GEV_0_Parameters_C0
constants
  GEV_0_diameter_min_val
  GEV_0_diameter_max_val
  GEV_0_CONTROL
  GEV_0_rate
axioms
  GEV_0_diameter_min_val = 0 // If position of a valve is at minimum, the valve is fully closed (0% open)
  GEV_0_diameter_max_val = 10 // On contrary, maximum means that the valve is fully open (100% open)
  GEV_0_CONTROL = {−1,0,1} // -1 - closing, 0 - OFF, 1 - opening
  GEV_0_rate = GEV_0_diameter_max_val // The rate showing how fast the valve opens
  theorem GEV_0_rate ≤ GEV_0_diameter_max_val − GEV_0_diameter_min_val
end
```

The instantiation also affects the machine of the component. Particularly, the state variables and the labels of the invariants, guards and actions have the same name with the sequence number as has been provided by the developer. For instance, the control and flow inputs of the general electro-valve are named GEV_0_control_I and GEV_0_flow_I, respectively. Similarly, the other variables specifying the output(s) (GEV_0_flow_O) and the internal state (position of the gate GEV_0_position and the mode GEV_0_mode) acquire the instantiated names. The properties of the component are added to the system specification in order to guarantee the correct operation of the component within the system. One can also have the deadlock freedom theorem to show that the component does not terminate. However, since this theorem is present in the library of components and there might be a need to show partial deadlock freedom at some refinement step, we omit it when the component is instantiated (Listing 28).

Listing 28. General electro-valve instantiated variables and properties

```
machine GEV_0_Behaviour_M0 sees GEV_0_Parameters_C0
variables
  GEV_0_control_I
  GEV_0_flow_I
  GEV_0_flow_O
  GEV_0_mode
  GEV_0_position
invariants
```

```
 GEV_0_control_I ∈ GEV_0_CONTROL ∧ GEV_0_mode ∈ 0..1 ∧
 GEV_0_flow_I ∈ GEV_0_diameter_min_val..GEV_0_diameter_max_val ∧
 GEV_0_flow_O ∈ GEV_0_diameter_min_val..GEV_0_diameter_max_val ∧
 GEV_0_position ∈ GEV_0_diameter_min_val..GEV_0_diameter_max_val ∧
 GEV_0_flow_O ≤ GEV_0_position ∧ (GEV_0_mode = 0 ⇒ GEV_0_flow_O ≤ GEV_0_flow_I)
```

At this moment, the system specification is as illustrated in Figure 8. The behaviour of the instantiated general electro-valve is the same as the parameterized elector-valve present in the library (see Section 4.2 in [22]). The instantiated component differs only in the precise values for the parameters and the names of the corresponding data structures due to the aforementioned instantiation process. Therefore, we omit the events modelling the behaviour.

4.2. First refinement: adding a connection to electro-valves controlling doors and gears

To develop the system further, we apply the refinement pattern presented in Section 3.1. This allows us to obtain the specification shown in Figure 9. The connector enables the introduction of the destination component in the subsequent refinement step.

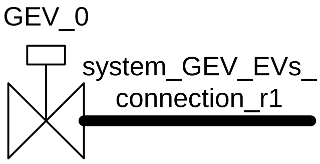

Figure 9. The general electro-valve with the connector

We start this refinement by defining a set of control values to be able to specify the order in which the execution will take place (Listing 29). Notice that the subscript numbering in the pattern is now transformed into specific numbers due to the naming conventions in the Rodin tool (e.g., R_{n-1} => R1).

Listing 29. Control set for the connection between the general electro-valve and the other valves

```
context GEV_0_Electrovalves_Connection_C1 extends GEV_0_Parameters_C0
constants SYSTEM_CONTROL_R1
axioms
 SYSTEM_CONTROL_R1 = {0,1,2}
end
```

Since we specify a connector and a control mechanism between the read of inputs by the general electro-valve and the update on the connector at this refinement step, we need to show that the newly introduced events terminate. Hence, we use a numeric variant (the value of the control variable `system_control_r1` whose value is decreased by the convergent events) as shown in Listing 30.

Listing 30. First refinement of the system: variables and properties

```
machine GEV_0_Electrovalves_Connection_M1 refines GEV_0_Behaviour_M0 sees GEV_0_Electrovalves_Connection_C1
variables
 ... // The variables derived from the GEV model
 system_control_r1
 system_GEV_0_EVs_connection_r1
invariants
 system_control_r1 ∈ SYSTEM_CONTROL_R1 ∧
 system_GEV_0_EVs_connection_r1 ∈ GEV_0_diameter_min_val..GEV_0_diameter_max_val
variant system_control_r1
```

Initially, the control variable is set to 0, so that the general electro-valve can start the execution sequence by reading its inputs. The value assigned to the connector is the minimum possible flow (Listing 31).

Listing 31. First refinement of the system: initialisation

```
event INITIALISATION extends INITIALISATION
  then
    system_control_r1 ≔ 0 || system_GEV_0_EVs_connection_r1 ≔ GEV_0_diameter_min_val
end
```

The control mechanism described in Section 3 affects the read of the inputs by the general electro-valve. Particularly, the guard of the environment event is strengthened (system_control_r1 = 0) and the set of actions is extended (system_control_r1 ≔ 1) following the pattern in Section 3.1 as illustrated in Listing 32.

Listing 32. First refinement of the system: modification of the general electro-valve environment event

```
event GEV_0_environment refines GEV_0_environment
  where
    GEV_0_mode = 0 ∧ system_control_r1 = 0
  then
    system_control_r1 ≔ 1 || GEV_0_mode ≔ 1 || GEV_0_control_I :∈ GEV_0_CONTROL ||
    GEV_0_flow_I :∈ GEV_0_diameter_min_val..GEV_0_diameter_max_val
end
```

Finally, we introduce an event (Listing 33) that models the connection between the general electro-valve and the specific electro-valves to be added in the later refinement steps. This event is as shown in pattern described in Section 3.1. Notice that this event is convergent and decreases the control variable system_control_r1 (i.e., the variant) from 1 to 0.

Listing 33. First refinement of the system: Introduction of event modelling connection

```
convergent event system_connection_GEV_0_EVs
  where
    GEV_0_mode = 0 ∧ system_control_r1 = 1
  then
    system_control_r1 ≔ 0 || system_GEV_0_EVs_connection_r1 ≔ GEV_0_flow_O
end
```

The second refinement step where we add the generic component as the destination follows the pattern presented in Section 3.2. We simply "drag-and-drop" the generic component from the formal library (see [22]) to the system specification. The overall graphical representation of the specification after two refinement steps is illustrated in Figure 10. The complete model of this refinement can be found in Appendix G. Due to obviousness of this refinement, we omit it and show the third refinement step.

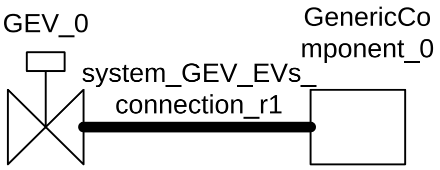

Figure 10. Graphical representation of the landing gear system after two refinements

4.3. Third refinement: refinement of the generic component into valves

In this refinement step, we derive the specification illustrated in Figure 11. The machine refinement of the generic component into a set of valves proceeds according to the pattern presented in Section 3.3 and the aforementioned instantiation rules. Thus, we only show the extra structures that can be used to make the proof of the refinement relation stronger (the complete model can be found in Appendix H).

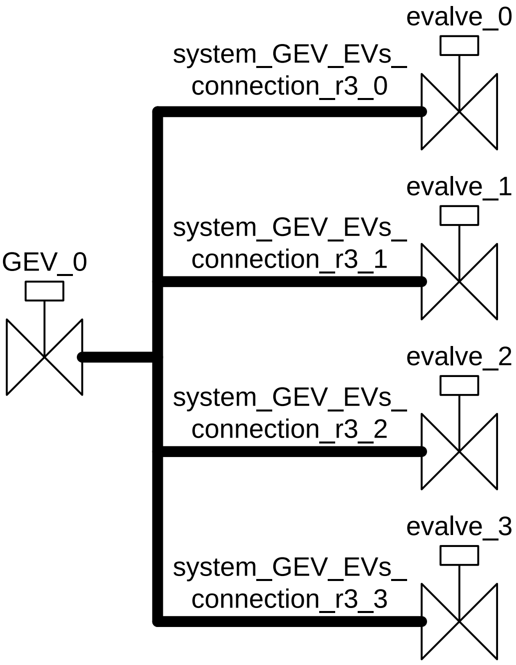

Figure 11. Landing gear system after three refinement steps

The context of this refinement step contains the set of the specific electro-valves, as well as exact values for their parameters (e.g., maximum diameter of electro-valve 0, evalve_0_diameter_max_val) as shown in Listing 34. In addition to the pattern data structures, we provide the theorems to show the compatibility between the general electro-valve and the specific electro-valves (e.g., theorem evalve_0_diameter_max_val = GEV_0_diameter_max_val). That is, all the valves should have the same diameters in order to be properly connected.

Listing 34. Instantiation of the parameters of the valves

```
context Electrovalves_Doors_Gears_C3 extends Electrovalves_Doors_Gears_Generic_C2
sets COMPONENTS_R3
constants evalve_0_r3
  evalve_1_r3
  evalve_2_r3
  evalve_3_r3
  evalve_0_diameter_min_val
  evalve_0_diameter_max_val
  evalve_0_CONTROL
  evalve_0_rate
  ... // Parameters of other valves
axioms
  partition(COMPONENTS_R3, {evalve_0_r3}, {evalve_1_r3}, {evalve_2_r3}, {evalve_3_r3}) ∧
  evalve_0_diameter_min_val = 0 ∧ evalve_0_diameter_max_val = 10 ∧ evalve_0_CONTROL = {−1,0,1} ∧
  evalve_0_rate = evalve_0_diameter_max_val ∧
  theorem evalve_0_rate ≤ evalve_0_diameter_max_val − evalve_0_diameter_min_val ∧
  ... /* Definitions of other valves */ ∧
  theorem evalve_0_diameter_max_val = GEV_0_diameter_max_val ∧
  theorem evalve_1_diameter_max_val = GEV_0_diameter_max_val ∧
  theorem evalve_2_diameter_max_val = GEV_0_diameter_max_val ∧
  theorem evalve_3_diameter_max_val = GEV_0_diameter_max_val
end
```

4.4. Fourth refinement: introduction of connections between the electro-valves and cylinders of doors

Once we have derived the specification shown in Figure 11, we can proceed with adding connectors and the cylinders. A cylinder requires two connections (see Section 4.3 in [22]): one for the head and the other one for the cap (Figure 7). The flows of these connections are controlled by the corresponding valves. For instance, the cylinders 0 to 2 operate according to the flow of the valves 0 and 1. Hence, we continue the development, so that we derive the specification whose graphical representation is shown in Figure 12.

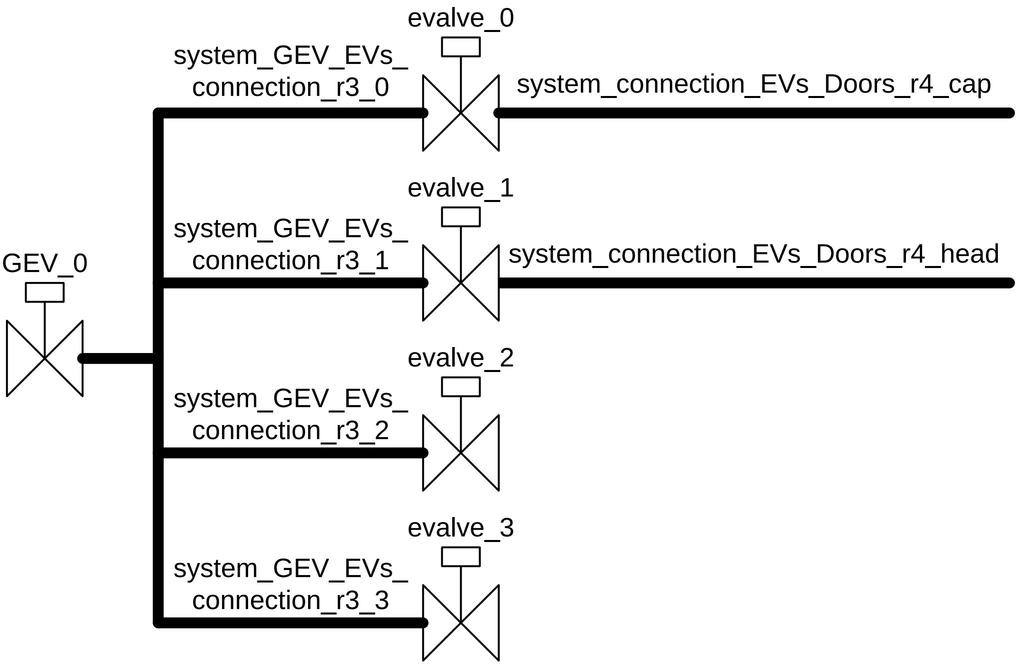

Figure 12. Visualisation of the landing gear specification after four refinements

Certainly, each connection can be introduced in separate refinement steps. However, since these steps are simple, we apply the connector pattern (Section 3.2) twice at the same step (see Appendix I for complete formal description). Therefore, we have two control sets for each connection (Listing 35).

Listing 35. Control sets for heads and caps of the cylinders

```
context EVs_Doors_Connection_C4 extends Electrovalves_Doors_Gears_C3
constants
  SYSTEM_CONTROL_R4_CAP
  SYSTEM_CONTROL_R4_HEAD
axioms
  SYSTEM_CONTROL_R4_CAP = {0,1,2} ∧ SYSTEM_CONTROL_R4_HEAD = {0,1,2}
end
```

Since the connection pattern is applied twice, we also need two connectors and two control variables. One control variable (system_control_r4_cap) determines the execution sequence between valve 0 and a connector for the caps of cylinders 0 to 2 (system_connection_EVs_Doors_r4_cap). The other control variable (system_control_r4_head) imposes the execution order between valve 1 and a connector for the heads of the same cylinders (system_connection_EVs_Doors_r4_head). Notice that a variant is a sum of the control values due to the fact that the pattern for introducing a connector has been applied twice (Listing 36).

Listing 36. Introduction of the connectors for heads and caps

```
machine M4_EVs_Doors_Connection refines M3_Electrovalves_Doors_Gears  sees C4_EVs_Doors_Connection
variables ...
  system_connection_EVs_Doors_r4_cap
  system_connection_EVs_Doors_r4_head
  system_control_r4_cap
  system_control_r4_head
invariants
  system_control_r4_cap ∈ SYSTEM_CONTROL_R4_CAP ∧ system_control_r4_head ∈ SYSTEM_CONTROL_R4_HEAD ∧
  system_connection_EVs_Doors_r4_cap ∈ evalve_0_diameter_min_val..evalve_0_diameter_max_val ∧
  system_connection_EVs_Doors_r4_head ∈ evalve_1_diameter_min_val..evalve_1_diameter_max_val
variant system_control_r4_cap + system_control_r4_head
```

Initially, the control variables are assigned the value 0, so that the execution sequence can start with the valves and proceed to the connectors. The values on the connectors are the minimums of the flows from the valves.

We now show the events that have been affected by the application of the pattern. Particularly, the environment events of valve 0 and valve 1 are refined considering the control variables as shown in Listing 37 and Listing 38, respectively.

Listing 37. Refinement of valve 0 environment event

```
  event evalve_0_environment extends evalve_0_environment
    where
      system_control_r4_cap = 0
    then
      system_control_r4_cap ≔ 1
  end
```

Listing 38. Refinement of valve 1 environment event

```
  event evalve_1_environment extends evalve_1_environment
    where
      system_control_r4_head = 0
    then
      system_control_r4_head ≔ 1
  end
```

In addition to these events, there are two newly introduced ones. These new events model the connections between valves 0 to 1 and the caps and heads of cylinders 0 to 2. Particularly, Listing 39 illustrates the value update of the connector for the caps whereas Listing 40 presents the value update of the connector for the heads. Notice that these events are convergent and each event decreases a corresponding control variable from 1 to 0.

Listing 39. Connection event between valve 0 and caps of the cylinders

```
  convergent event system_connection_EVs_Doors_cap
    where   evalve_0_mode = 0 ∧ system_control_r4_cap = 1
    then   system_control_r4_cap ≔ 0 || system_connection_EVs_Doors_r4_cap ≔ evalve_0_flow_O
  end
```

Listing 40. Connection event between valve 1 and head of the cylinders

```
  convergent event system_connection_EVs_Doors_head
    where   evalve_1_mode = 0 ∧ system_control_r4_head = 1
    then   system_control_r4_head ≔ 0 || system_connection_EVs_Doors_r4_head ≔ evalve_1_flow_O
  end
```

4.5. Fifth refinement: introduction of cylinders without generic component

After the connectors are present in the system, we can extend the specification of the system with the cylinders 0 to 2. We add them at the same step as they operate simultaneously according to the flows from valves 0 and 1. The system derived to this point whose complete specification can be found in Appendix J is visualised in Figure 13.

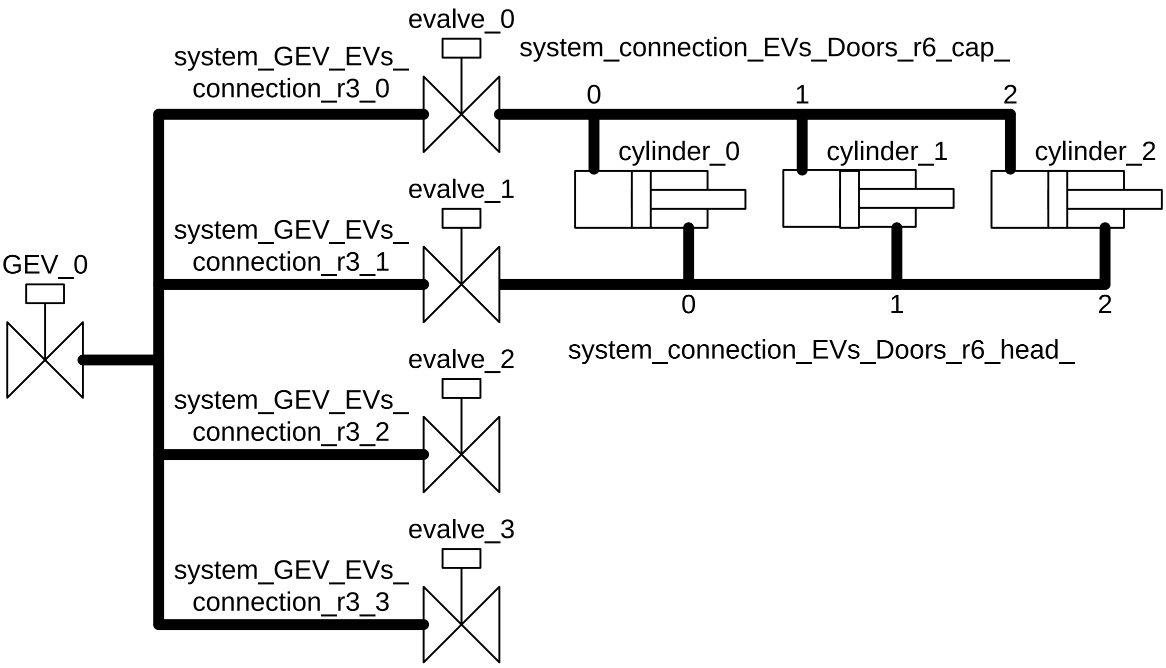

Figure 13. Landing gear system after five refinement steps

According to the pattern for introducing components without the generic one, the context of this refinement contains the set of cylinders. It also includes the parameters of the cylinders instantiated with specific values (e.g., the maximum diameter of the inputs of cylinder 0 `cylinder_0_input_diameter_max_val`). In addition, there are theorems (e.g., `theorem cylinder_0_input_diameter_max_val = evalve_0_diameter_max_val`) to show that the cylinders are compatible with the valves (Listing 41).

Listing 41. Definitions of the cylinders 0 to 2

```
context Cylinders_Doors_C5 extends EVs_Doors_Connection_C4
sets COMPONENTS_R5
constants
  SYSTEM_CONTROL_R5_CAP
  SYSTEM_CONTROL_R5_HEAD
  cylinder_0_r5
  cylinder_1_r5
  cylinder_2_r5
  cylinder_0_input_diameter_min_val
  cylinder_0_input_diameter_max_val
  cylinder_0_cap_pos
  cylinder_0_head_pos
  ... // Parameters of other cylinders
axioms
  SYSTEM_CONTROL_R5_CAP = {0,1,2} ∧ SYSTEM_CONTROL_R5_HEAD = {0,1,2} ∧
  partition(COMPONENTS_R5, {cylinder_0_r5}, {cylinder_1_r5}, {cylinder_2_r5}) ∧
  // 0 stands for no liquid flowing into the cylinder (0% open)
  cylinder_0_input_diameter_min_val = 0 ∧
  // 100 stands for maximum velocity the piston can move inside the cylinder (100% open)
```

```
  cylinder_0_input_diameter_max_val = 10 ∧
  cylinder_0_cap_pos = 0 ∧ cylinder_0_head_pos ∈ ℕ1 ∧
  … /* Parameters of other cylinders */ ∧
  theorem cylinder_0_input_diameter_max_val = evalve_0_diameter_max_val ∧
  theorem cylinder_1_input_diameter_max_val = evalve_0_diameter_max_val ∧
  theorem cylinder_2_input_diameter_max_val = evalve_0_diameter_max_val
end
```

The machine refinement follows the pattern presented in Section 3.4. Particularly, each cylinder has a couple of its own connectors: one for the cap (e.g., system_connection_EVs_Doors_r5_cap_0) and the other one for the head (e.g., system_connection_EVs_Doors_r5_head). There are two control variables that provide the execution sequence between the valve 0 and caps (system_control_r5_cap) as well as between valve 1 and heads (system_control_r5_head) of the cylinders. These variables data refine the old control variables, namely system_control_r4_cap and system_control_r4_head, respectively. Finally, the variable cylinders_read_r5 is introduced to manage the reads of the inputs by the cylinders (Listing 42).

Listing 42. Application of the pattern from Section 3.4

```
machine M5_Cylinders_Doors refines M4_EVs_Doors_Connection  sees C5_Cylinders_Doors
variables
  system_connection_EVs_Doors_r5_cap_0
  system_connection_EVs_Doors_r5_head
  system_control_r5_cap
  system_control_r5_head
  cylinders_read_r5
  cylinder_0_piston_position_O
  cylinder_0_flow_cap_I
  cylinder_0_flow_head_I
  cylinder_0_mode
  … // Variables of the cylinders and other connectors
invariants
  // The mode and the current position of the piston in the cylinder
  cylinder_0_mode ∈ 0..1 ∧ cylinder_0_piston_position_O ∈ cylinder_0_cap_pos..cylinder_0_head_pos ∧
  // Input to move the piston to the right
  cylinder_0_flow_cap_I ∈ cylinder_0_input_diameter_min_val..cylinder_0_input_diameter_max_val ∧
  // Input to move the piston to the left
  cylinder_0_flow_head_I ∈ cylinder_0_input_diameter_min_val..cylinder_0_input_diameter_max_val ∧
  system_connection_EVs_Doors_r5_cap_0 ∈ cylinder_0_input_diameter_min_val..cylinder_0_input_diameter_max_val ∧
  system_connection_EVs_Doors_r5_head_0 ∈ cylinder_0_input_diameter_min_val..cylinder_0_input_diameter_max_val ∧
  system_connection_EVs_Doors_r4_cap = system_connection_EVs_Doors_r5_cap_0 ∧ … ∧
  system_connection_EVs_Doors_r4_head = system_connection_EVs_Doors_r5_head_0 ∧ … ∧
  (system_control_r4_cap = 0 ⇔ system_control_r5_cap = 0 ∨ system_control_r5_cap = 2)
  (system_control_r4_cap = 1 ⇔ system_control_r5_cap = 1) ∧
  (system_control_r4_head = 1 ⇔ system_control_r5_head = 1) ∧
  (system_control_r4_head = 0 ⇔ system_control_r5_head = 0 ∨ system_control_r5_head = 2) ∧
  cylinders_read_r5 ∈ COMPONENTS_R5 → 0..1
variant card(COMPONENTS_R5) − cylinders_read_r5(cylinder_0_r5) − cylinders_read_r5(cylinder_1_r5) –
                                                                            cylinders_read_r5(cylinder_2_r5)
```

The pattern affects the refinement of the environment events of the valves. For instance, Listing 43 shows the refinement of the environment event of valve 0 which controls the flow of liquid to the caps of the cylinders using the variable system_control_r5_cap. The environment event of valve 1 is refined similarly (see Appendix J for the details).

Listing 43. Refinement of valve 0 environment event

```
event evalve_0_environment refines evalve_0_environment
  where
    evalve_0_mode = 0 ∧ system_control_r2 = 2 ∧ valves_read_r3(evalve_0_r3) = 0 ∧ system_control_r5_cap = 0
  then
    evalve_0_mode ≔ 1 || evalve_0_control_I :∈ evalve_0_CONTROL || system_control_r5_cap ≔ 1 ||
    evalve_0_flow_I ≔ system_GEV_0_EVs_connection_r3_0 || valves_read_r3(evalve_0_r3) ≔ 1
end
```

The connection events update the connectors (e.g., system_connection_EVs_Doors_r5_cap_0 ≔ evalve_0_flow_O) as well as reset the control variable cylinders_read_r5 (cylinders_read_r5 ≔ COMPONENTS_R5 × {0}). An example of the connection event for caps is shown in Listing 44.

Listing 44. Refinement of the connection event between valve 0 and caps of cylinders 0 to 2

```
event system_connection_EVs_Doors_cap refines system_connection_EVs_Doors_cap
  where
    evalve_0_mode = 0 ∧ system_control_r5_cap = 1
  then
    system_control_r5_cap ≔ 2 || cylinders_read_r5 ≔ COMPONENTS_R5 × {0} ||
    system_connection_EVs_Doors_r5_cap_0 ≔ evalve_0_flow_O ||
    system_connection_EVs_Doors_r5_cap_1 ≔ evalve_0_flow_O ||
    system_connection_EVs_Doors_r5_cap_2 ≔ evalve_0_flow_O
end
```

Finally, we introduce the environment events of the cylinders considering the pattern. Listing 45 illustrates an environment event of cylinder 0. Since the cylinder operates only after both inputs are read, the guards contain the check of the both control variables (system_control_r5_cap = 2 ∧ system_control_r5_head = 2). The new values for the control variables are also computed simultaneously (e.g., cylinders_read_r5[COMPONENTS_R5∖{cylinder_0_r5}] = {1} ⇒ *system_control_cap_new_r5* = 0 ∧ *system_control_head_new_r5* = 0). The environment events of the other cylinders are added in the same manner.

Listing 45. Introduction of the environment event of cylinder 0

```
convergent event cylinder_0_environment
  any system_control_cap_new_r5 system_control_head_new_r5
  where
    cylinder_0_mode = 0 ∧ system_control_r5_cap = 2 ∧ system_control_r5_head = 2 ∧
    (cylinders_read_r5[COMPONENTS_R5∖{cylinder_0_r5}] = {1} ⇒ system_control_cap_new_r5 = 0 ∧
                                                        system_control_head_new_r5 = 0) ∧
    (¬cylinders_read_r5[COMPONENTS_R5∖{cylinder_0_r5}] = {1} ⇒ system_control_cap_new_r5 = 2 ∧
                                                        system_control_head_new_r5 = 2) ∧
    cylinders_read_r5(cylinder_0_r5) = 0
  then
    cylinder_0_mode ≔ 1 || cylinder_0_flow_cap_I ≔ system_connection_EVs_Doors_r5_cap_0 ||
    cylinder_0_flow_head_I ≔ system_connection_EVs_Doors_r5_head_0 ||
    system_control_r5_cap ≔ system_control_cap_new_r5 || system_control_r5_head ≔ system_control_head_new_r5 ||
    cylinders_read_r5(cylinder_0_r5) ≔ 1
end
```

4.6. Case study summary

The system development proceeds in the similar manner by applying the corresponding patterns when necessary until all the components are introduced. We completely developed the part of the case study comprised the valves and the cylinders by applying different patterns in the proposed manner. The summary of the proof statistics for the case study is shown in Table 1. Most proof obligations were proven by the tool.

A large number of the manual proof obligations were derived from the specifications of the components and can be simply copied from the library by the tool. Notice that the second group of cylinders (3 to 5) has been introduced by using the generic component in between. This is just to exemplify the scalability and reusability of our approach.

Table 1. Case study proof statistics

Ref. n/n	Name	Total POs	Auto
0	General electro-valve	24	21
1	Connection between general electro-valve and the other valves	7	7
2	Generic component for electro-valves of doors and gears	44	43
3	Electro-valves of doors and gears	142	125
4	Connection between electro-valves of doors and cylinders of doors	14	14
5	Cylinders of doors	87	84
6	Connection between electro-valves of gears and cylinders of gears	14	13
7	Generic component for cylinders of gears	59	57
8	Cylinders of gears	68	55
	Summary	459	419

5. Related Work

BMotionStudio has been proposed as an approach to visual simulation of the Event-B models [3][4]. The idea behind BMotionStudio is that the designer creates a domain specific image and links the model to it using a gluing code written in JavaScripts. The simulation is based on the ProB animator and model checker [5], so that whenever the model is executed the corresponding graphical element reacts on the changes. The BMotionStudio tool also supports interaction with the user, i.e., a user can provide an input through visual elements instead of manipulating the model directly.

Instead of visualizing the execution of the already developed model, we propose to build Event-B model in a visual manner. We rely on the formal library of parameterized visual components available at the developer’s disposal. The development of the specification is then a process of the instantiation of the necessary components and the connection of them into a system. That is, the developer does not need to redraw the graphical representation but to simply reuse (instantiate) the components. Eventually, the designer obtains a graphical representation of the system whilst its specification is in fact written in Event-B with correctness proof. Certainly, our approach can be complemented by the BMotionStudio in order to obtain visualisation of the model execution.

Snook and Butler [6] have proposed an approach to merge visual UML [7] that lacks formal precise semantics with B [8] that requires significant effort in training to overcome the mathematical barrier. This approach has then been extended to Event-B and called iUML-B [9]. The authors define semantics of UML by translating it to Event-B. The use of UML-B profile provides specialisation of UML entities to support refinement. The authors also present the tools that automatically generate an Event-B model from a UML one.

In contrast to using UML as visualisation tool, we aim to enhance scalability of the Event-B development by utilising visual components from a formal library (see [22] for examples). The aim is to facilitate the rigorous development process by visual design, where the developers pick the necessary components, instantiate them and connect according to the refinement pattern proposed in this paper. The system specification is then a visual model that comprises a composition of the instantiated versions of these components. Nevertheless, we target automated generation of the necessary data structures and Event-B elements whenever our approach is applied.

Edmunds, Waldén and Snook have proposed an approach towards component-based reuse for Event-B [10]. The proposed approach is based on an extension to iUML-B class diagrams [9] and an extension to the shared-event composition technique proposed in [11]. The authors propose a notation for the local parameters of events in order to facilitate event composition when connecting components. The notation is to reveal communicating parameters that form the interfaces of the events to synchronize. The authors consider composition invariants that specify the properties about the composition. A composition machine then includes the constituent machines. In addition, the authors propose to apply design-by-contract [12] in order to ensure correct connection, i.e., that the values of the corresponding inputs and outputs are within the allowable range.

In contrast to the approach described in [10], we consider components as parameterised (generic) Event-B specifications, each of which is assigned a specific visual symbol. The designer is then instantiates the necessary components and connects them into a system by the use of the proposed refinement patterns. Our goal is to facilitate rigorous development in Event-B by visual design, i.e., to improve human-machine interface and communication between the developer and the customer. We aim to provide the developers with “drag-and-drop” approach, where the components are picked from the library, instantiated and connected into a system in a graphical fashion. Nevertheless, we propose to use the inclusion mechanism similarly to [10], so that the instantiated specifications of the components are included into a system specification. We may also adopt the design-by-contract approach in order to stronger support the proper connectivity between the components.

An approach to a component-based formal design within Event-B has been proposed by Ostroumov, Tsiopoulos, Plosila and Sere [13]. The aim of this work is the generation of a structural VHDL [14] description from a formal model. The authors present a one-to-one mapping of formal functions defined in an Event-B context and library components derived from VHDL. Using this mapping, the authors rely on an additional refinement step, in which regular operations are replaced with function calls. This allows for automated generation of structural VHDL descriptions.

In contrast to this approach, we propose an approach to systems development in Event-B in a visual manner. This approach is not limited to VHDL descriptions and allows the designers to utilize various components from different application domains. We present the refinement patterns to enable composition of the necessary components into a system in a visual systematic manner, so that the developers can build the system in a “drag-and-drop” manner.

A modularization mechanism to support scalability of Event-B modelling has been proposed by Iliasov et al. [15]. The authors consider sequential systems whose functionality is distributed among several components. The authors propose to extend the language of Event-B with (atomic) operation calls and introduce the notion of modules (i.e., components) which contain groups of callable operations. The modules can have internal and external states and invariants that express properties on these states. According to the authors, their approach can be seen as a special type of the A-style decomposition approach proposed by Abrial [16]. The goal is to split a monolithic model into sub-models, each of which can be further developed separately in parallel. However, once all the modules contain the necessary level of detail, they can be composed back into a system. The composition mechanism is supported by the corresponding proofs.

Instead of extending the Event-B language and decomposing the system into components and composing it back out of modules, we propose to utilize a formal library of predefined parameterized components, each of which has a corresponding visual symbol. The components form a system and can be connected sequentially; however, they can process the input data in parallel. We show the refinement patterns which provide the connection mechanism between the components following the refinement approach. That is, we propose to merge top-down and bottom-up development approaches in order to enhance scalability of rigorous development in Event-B. Our goal is to enable graphical system development in Event-B whilst preserving its advantages in terms of the rigour and correctness proof mechanism.

6. Conclusion and future work

We proposed a systematic approach to the visual system development in Event-B. We rely on the formal library of parameterized visual components, out of which the developer can pick and instantiate the necessary components according to the requirements. These components are connected using the proposed composition approach and a set of the refinement patterns. They enable seamless composition (integration) of various components into a system, where the visual layer is built on top of the formal one. The visual layer facilitates scalability and reusability by merging the top-down (refinement) and bottom-up (components) approaches. The approach is also flexible, so that the developer can add more properties using the usual Event-B approach, if needed.

During the system development, one has to show that the connection between the components is feasible and well-defined. This can be done by applying the design-by-contract mechanism [12] that was mentioned earlier. Thus, one direction of our future work is to investigate this issue.

Due to the systematic nature of the proposed approach, it can be implemented in the form of tool, namely a plug-in to the Rodin platform. This will ease the use of the components and composition refinement patterns. Therefore, another future direction is towards tool development.

Acknowledgment

The authors would like to thank Dr. Marta Olszewska and Dr. Andrew Edmunds for the fruitful discussions on the topic of this paper.

References

[1] J.-R. Abrial, Modeling in Event-B: System and Software Engineering. Cambridge: Cambridge University Press, 2010.

[2] R. J. Back and J. Wright, Refinement Calculus: A Systematic Introduction, Springer-Verlag, 1998.

[3] L. Ladenberger, J. Bendisposto, M. Leuschel, Visualising Event-B Models with B-Motion Studio, Proceedings of Formal Methods for Industrial Critical Systems (FMICS), LNCS: Springer Berlin Heidelberg, pp. 202-204, 2009.

[4] BMotion Studio for ProB Handbook, 2015. Available: http://nightly.cobra.cs.uni-duesseldorf.de/bmotion/bmotion-prob-handbook/nightly/html/index.html. Visualising Event-B Models with B-Motion Studio

[5] M. Leuschel, M. Butler, ProB: A Model Checker for B, Proc. FME, Springer, vol. 2805, 2003, p. 855-874.

[6] C. Snook, M. Butler, UML-B: Formal Modeling and Design Aided by UML, ACM Transactions on Software Engineering and Methodology, Vol. 15(1), pp. 92–122, 2006.

[7] G. Booch, I. Jacobson, J. Rumbaugh, The unified modeling language – a reference manual, 2nd edition, Addison-Wesley, p. 721, 2004.

[8] S. Schneider, The B-method: An Introduction, Basingstoke: Palgrave, p. 370, 2001.

[9] C. Snook, M. Butler, UML-B and Event-B: an integration of languages and tools, Proceedings of IASTED International Conference on Software Engineering, pp. 12, 2008.

[10] A. Edmunds, M. Waldén, C. Snook, Towards Component-based Reuse for Event-B, Proceedings of 27th Nordic Workshop on Programming Theory, Reykjavik University, Iceland pp. 3, 2015.

[11] R. Silva, Supporting Development of Event-B Models, PhD thesis, University of Southampton, 2012.

[12] B. Meyer, Design by Contract: The Eiffel Method, In TOOLS (26), IEEE, p. 446, 1998.

[13] S. Ostroumov, L. Tsiopoulos, J. Plosila, K. Sere, Generation of Structural VHDL Code with Library Components From Formal Event-B Models, In Proceedings of Euromicro Conference on Digital System Design, IEEE Conference Publishing Services (CPS), pp. 111-118, 2013.

[14] IEEE Standard VHDL Language Reference Manual, IEEE 1076, 2008.

[15] A. Iliasov, E. Troubitsyna, L. Laibinis, A. Romanovsky, K. Varpaaniemi, D. Ilic, T. Latvala, Supporting Reuse in Event B Development: Modularisation Approach, In Proceedings of Abstract State Machines, Alloy, B, and Z (ABZ), pp. 17, 2010.

[16] Event Model Decomposition, J.-R. Abrial, 2009. Available: http://wiki.event-b.org/images/Event_Model_Decomposition-1.3.pdf.

[17] M. Butler, E. Sekerinski, K. Sere, An Action System Approach to the Steam Boiler Problem, Formal Methods For Industrial Applications, Vol. 1165, LNCS: Springer-Verlag, pp. 129-148, 1996.

[18] RODIN, 2014. Available: http://sourceforge.net/projects/rodin-b-sharp/.

[19] C. Métayer, J.-R. Abrial, L. Voisin, Deliverables, Rigorous Open Development Environment for Complex Systems, 2005. Available: http://rodin.cs.ncl.ac.uk/deliverables/D7.pdf.

[20] K. Robinson, System Modelling & Designing using Event-B, 2010. Available: http://wiki.event-b.org/images/SM%26D-KAR.pdf.

[21] ABZ 2014: The Landing Gear Case Study, Communications in Computer and Information Science 433, Edited by F. Boniol, V. Wiels, Y. A. Ameur, K.-D. Schewe, Springer, pp. 171, 2014.

[22] S. Ostroumov, M. Waldén, Formal Library of Visual Components, TUCS Technical Report Series, No. 1147, pp. 34, 2015.

Appendix A

The complete model of the connector pattern

```
context System_Connection_C_{n-1} extends Component_i_C_{n-2}
constants SYSTEM_CONTROL_R_{n-1}
axioms
  @system_axm_r_{n-1}_0 SYSTEM_CONTROL_R_{n-1} = {0,1,2}
end
```

```
machine System_Connection_M_{n-1} refines Component_i_M_{n-2} sees System_Connection_C_{n-1}
variables ...
  system_control_r_{n-1}
  system_connection_Component_i_Component_k_r_{n-1}
invariants
  @system_control_r_{n-1} system_control_r_{n-1} ∈ SYSTEM_CONTROL_R_{n-1}
  @system_connection_Component_i_Component_k_r_{n-1}
    system_connection_Component_i_Component_k_r_{n-1} ∈ <COMPONENT_i_OUTPUT_TYPE>
variant system_control_r_{n-1}
events
  event INITIALISATION
  extends INITIALISATION
    then
      @system_act_r_{n-1}_0 system_control_r_{n-1} ≔ 0
      @system_act_r_{n-1}_1 system_connection_Component_i_Component_k_r_{n-1} ≔ <INIT_VALUE>
  end

  event Component_i_environment
  refines Component_i_environment
    where
      ... // Other guards derived from the component i
      @system_grd_r_{n-1}_0 system_control_r_{n-1} = 0
    then
      ... // Other actions derived from the component i
      @system_act_r_{n-1}_0 system_control_r_{n-1} ≔ 1
  end

  convergent event system_connection_Component_i_Component_k
    where
      @system_grd_r_{n-1}_0 system_control_r_{n-1} = 1
       // We need to be sure that the component i has updated its outputs
      // The component mode can also be of Boolean type (0 ⇔ FALSE, 1 ⇔ TRUE), if there are only two alternating modes
      @system_grd_r_{n-1}_1 <Component_i_mode> = 0
    then
      @system_act_r_{n-1}_0 system_control_r_{n-1} ≔ 0
      @system_act_r_{n-1}_1 system_connection_Component_i_Component_k_r_{n-1} ≔ <Component_i_Output>
  end
  ...
end
```

Appendix B

The complete model of the component introduction pattern

context Component_k_Parameters_C_n **extends** System_Connection_C_{n-1}
constants SYSTEM_CONTROL_R_n
axioms
… *// Parameters of component k, if any*
@system_axm_r_n_0 SYSTEM_CONTROL_R_n = {0,1,2}
end

machine Component_k_M_n **refines** System_Connection_M_{n-1} **sees** Component_k_Parameters_C_n
variables …
system_connection_Component_i_Component_k_r_{n-1}
~~system_control_r_{n-1}~~
GenericComponent_k_I
GenericComponent_k_O
GenericComponent_k_mode
GenericComponent_k_IOrelation
system_control_r_n
invariants
… *// The types and the properties of the generic component*
@system_control_r_n_0 system_control_r_n $\in$ SYSTEM_CONTROL_R_n
@system_glueinv_r_n_1 system_control_r_{n-1} = 0 $\Leftrightarrow$ system_control_r_n = 0 $\lor$ system_control_r_n = 2
@system_glueinv_r_n_2 system_control_r_{n-1} = 1 $\Leftrightarrow$ system_control_r_n = 1
variant system_control_r_n
events
event INITIALISATION
then
…
@system_act_r_{n-1}_1 system_connection_Component_i_Component_k_r_{n-1} ≔ **<INIT_VALUE>**
@GenericComponent_act_0 GenericComponent_k_mode ≔ 0
@GenericComponent_act_1 GenericComponent_k_I, GenericComponent_k_O, GenericComponent_k_IOrelation :|
GenericComponent_I' $\in$ {i | i $\in$ $\mathbb{P}1(\mathbb{Z})$ $\land$ finite(i)} $\land$
GenericComponent_O' $\in$ {o | o $\in$ $\mathbb{P}1(\mathbb{Z})$ $\land$ finite(o)} $\land$
GenericComponent_IOrelation' $\in$ GenericComponent_I' $\leftrightarrow$ GenericComponent_O' $\land$
dom(GenericComponent_IOrelation') = GenericComponent_I' $\land$
ran(GenericComponent_IOrelation') = GenericComponent_O'
@system_act_r_n_0 system_control_r_n ≔ 0
end

event Component_i_environment **refines** Component_i_environment
where
… *// Other guards derived from the component i*
~~@system_grd_r_{n-1}_0 system_control_r_{n-1} = 0~~
@system_grd_r_n_0 system_control_r_n = 0
then
… *// Other actions derived from the component i*
~~@system_act_r_{n-1}_0 system_control_r_{n-1} ≔ 1~~
@system_act_r_n_0 system_control_r_n ≔ 1
end

event system_connection_Component_i_Component_k **refines** system_connection_Component_i_Component_k
where

```
     @system_grd_$r_{n-1}$_1 <Component_i_mode> = 0
     @system_grd_$r_n$_0 system_control_$r_n$ = 1
    then
     @system_act_$r_{n-1}$_1 system_connection_Component_i_Component_k_$r_{n-1}$ ≔ <Component_i_Output_Value>
     @system_act_$r_n$_0 system_control_$r_n$ ≔ 2

   convergent event GenericComponent_k_environment
    where
     @grd_0 GenericComponent_k_mode = 0
     @system_grd_$r_n$_0 system_control_$r_n$ = 2
    then
     @act_0 GenericComponent_k_mode ≔ 1
     @act_1 GenericComponent_k_I ≔ <SET_OF_OUTPUT_VALUES_OF_COMPONENT_i>
     @system_act_$r_n$_0 system_control_$r_n$ ≔ 0
   end
   …
  end
```

Appendix C

Refinement pattern: Generic component into a set of specific ones

context Specific_Components_Parameters_C_{n+1} **extends** Component_k_Parameters_C_n
sets COMPONENTS_R_{n+1}
constants ...
  component_0_r_{n+1}
  ... //
  component_j_r_{n+1}
axioms
  ... *// Other parameters of the components*
  @system_components_set_r_{n+1} partition(COMPONENTS_R_{n+1}, {component_0_r_{n+1} }, ..., {component_j_r_{n+1}})
end

machine Specific_Components_M_{n+1} **refines** Component_k_M_n **sees** Specific_Components_Parameters_C_{n+1}
variables ...
  ~~GenericComponent_k_I~~
  ~~GenericComponent_k_O~~
  ~~GenericComponent_k_IOrelation~~
  ~~system_connection_Component_i_Component_k_r_{n-1}~~
  GenericComponent_k_mode
  system_connection_i_0_r_{n+1}
  ... *// Similar connectors to other components*
  system _connection_i_j_r_{n+1}
  components_read_r_{n+1}
invariants
  ... *// The properties of the specific components*
  @system_connection_i_0_r_{n+1}_0 system_connection_i_0_r_{n+1} ∈ <COMPONENT_0_INPUT_TYPE>
  @system_connection_i_j_r_{n+1}_1 system_connection_i_j_r_{n+1} ∈ <COMPONENT_j_INPUT_TYPE>
  @system_components_read_r_{n+1}_2 components_read_r_{n+1} ∈ COMPONENTS_R_{n+1} → 0..1
  @system_glueinv_r_{n+1}_10 GenericComponent_k_I ⊆ $\bigcup^{m}_{i=0}$INPUT_TYPE$_i$
  @system_glueinv_r_{n+1}_11 GenericComponent_k_O ⊆ $\bigcup^{m}_{i=0}$OUTPUT_TYPE$_i$
  @system_glueinv_r_{n+1}_12 GenericComponent_k_IOrelation ⊆ $\bigcup^{m}_{i=0}$INPUT_TYPE$_i$ × $\bigcup^{m}_{i=0}$OUTPUT_TYPE$_i$
  @system_inv_r_{n+1}_13 system_control_rn = 0 ∧ GenericComponent_k_mode = 1 ⇒
      components_read_r_{n+1} [COMPONENTS_R_{n+1}] = {0}
  @system_connection_i_0_r_{n+1}_14 system_connection_Component_i_Component_k_r_{n-1} = system_connection_i_0_r_{n+1}
  @system_connection_i_j_r_{n+1}_15 system_connection_Component_i_Component_k_r_{n-1} = system_connection_i_j_r_{n+1}
  @system_inv_r_{n+1}_9 system_control_r_n = 0 ∧ GenericComponent_k_mode = 1 ⇒
      components_read_r_{n+1}[COMPONENTS_R_{n+1}] = {0}
variant card(COMPONENTS_R_{n+1}) – components_read_r_{n+1}(component_0_r_{n+1}) – ... –
      components_read_r_{n+1}(component_j_r_{n+1})

events
  event INITIALISATION
   with
    @GenericComponent_k_I' GenericComponent_k_I' = $\bigcup^{m}_{i=0}$INPUT_TYPE$_i$
    @GenericComponent_k_O' GenericComponent_k_O' = $\bigcup^{m}_{i=0}$OUTPUT_TYPE$_i$
    @GenericComponent_k_IOrelation' GenericComponent_k_IOrelation' = $\bigcup^{m}_{i=0}$INPUT_TYPE$_i$ × $\bigcup^{m}_{i=0}$OUTPUT_TYPE$_i$
   then

```
  ... // Initialization of other state variables from previous refinements
  @system_act_r_{n+1}_0 system_connection_i_0_r_{n+1} ≔ <INIT_VALUE>
  @system_act_r_{n+1}_j system_connection_i_j_r_{n+1} ≔ <INIT_VALUE>
  @system_components_read_r_{n+1} components_read_r_{n+1} ≔ COMPONENTS_R_{n+1} × {0}
end

event system_connection_i_0j refines system_connection_Component_i_Component_k
 where
  @system_grd_r_{n-1}_1 <Component_i_mode> = 0
  @system_grd_r_n_0 system_control_r_n = 1
 then
  @system_act_r_n_0 system_control_r_n ≔ 2
  @system_act_r_{n+1}_0 system_connection_i_0_r_{n+1} ≔ <Component_i_Output>
  @system_act_r_{n+1}_j system_connection_i_j_r_{n+1} ≔ <Component_i_Output>
end

convergent event Component_0_environment
 where
  @grd0_0 component_0_mode = 0
  @system_grd_r_{n+1}_0 system_control_r_n = 2
  @system_components_read_grd_r_{n+1} components_read_r_{n+1}(component_0_r_{n+1} ) = 0
 then
  @act0_0 component_0_mode ≔ 1
  @act0_1 component_0_I_0 ≔ system_connection_i_0_r_{n+1}
  ... // Update of the other inputs not connected to component i, if any
  @system_components_read_act_r_{n+1} components_read_r_{n+1}(component_0_r_{n+1} ) ≔ 1
end

convergent event Component_j_environment
 where
  @grd0_0 component_j_mode = 0
  @system_grd_r_{n+1}_0 system_control_r_n = 2
  @system_components_read_grd_r_{n+1} components_read_r_{n+1}(component_j_r_{n+1} ) = 0
 then
  @act0_0 component_j_mode ≔ 1
  @act0_1 component_j_I_0 ≔ system_connection_i_j_r_{n+1}
  ... // Update of the other inputs not connected to component i, if any
  @system_components_read_act_r_{n+1} components_read_r_{n+1}(component_j_r_{n+1} ) ≔ 1
end

event GenericComponent_k_environment refines GenericComponent_k_environment
 where
  @grd0_0 GenericComponent_k_mode = 0
  @system_grd_r_n_0 system_control_r_n = 2
  @system_grd_r_{n+1}_0 components_read_r_{n+1}[COMPONENTS_R_{n+1}] = {1}
 then
  @act0_0 GenericComponent_k_mode ≔ 1
  @system_act_r_n_0 system_control_r_n ≔ 0
  @system_act_r_{n+1}_0 components_read_r_{n+1} ≔ COMPONENTS_R_{n+1} × {0}
end
```

Appendix D

Refinement pattern: introduction of specific components without generic one

```
context Specific_components_Cn extends System_connection_Cn-1
sets COMPONENTS_Rn
constants
  SYSTEM_CONTROL_Rn
  component_0_rn
  … // Other components: component_q_rn, q ∈ 0..j
  component_j_rn
  … // Parameters of the components
axioms
  @system_axm_rn_1 SYSTEM_CONTROL_Rn = {0,1,2}
  @system_evalve_set partition(COMPONENTS_Rn, {component_0_rn}, …, {component_j_rn})
  … // Definitions of the components
end



machine Specific_Components_Mn refines System_Connection_Mn-1 sees Specific_components_Cn
variables
  system_control_rn
  components_read_rn
  system_connection_i_0_rn
  system_connection_i_j_rn
  … // State variables of the components and other connectors
invariants
  … // Properties of the components
  @system_components_read_rn_0 components_read_rn ∈ COMPONENTS_Rn → 0..1
  @system_control_rn_1 system_control_rn ∈ SYSTEM_CONTROL_Rn
  @system_connection_i_0_rn_2 system_connection_i_0_rn ∈ <COMPONENT_0_INPUT_TYPE>
  @system_connection_i_j_rn_3 system_connection_i_j_rn ∈ <COMPONENT_j_INPUT_TYPE>
  @system_glueinv_rn_4 system_connection_i_k_rn-1 = system_connection_i_0_rn
  @system_glueinv_rn_5 system_connection_i_k_rn-1 = system_connection_i_j_rn
  @system_glueinv_rn_6 system_control_rn-1 = 0 ⇔ system_control_rn = 0 ∨ system_control_rn = 2
  @system_glueinv_rn_7 system_control_rn-1 = 1 ⇔ system_control_rn = 1
variant card(COMPONENTS_Rn) − components_read_rn(component_0_rn) − … − components_read_rn(component_j_rn)
events
  event INITIALISATION
    then
      …
      @system_act_rn_0 system_control_rn ≔ 0
      @system_act_rn_1 components_read_rn ≔ COMPONENTS_Rn × {0}
      @system_act_rn_2 system_connection_i_0_rn ≔ <INIT_VALUE>
      … // Initialization of other connectors
      @system_act_rn_3 system_connection_i_j_rn ≔ <INIT_VALUE>
  end

  event Component_i_environment refines Component_i_environment
    where
      @grd0_0 <Component_i_mode> = 0
      @system_grd_rn_0 system_control_rn = 0
    then
      @act0_0 <Component_i_mode> ≔ 1
      @system_act_rn_0 system_control_rn ≔ 1
```

```
      ... // Read the new input
    end

    event system_connection_i_0...j refines system_connection_Component_i_Component_k
      where
        @system_grd_rₙ₋₁_0 <Component_i_mode> = 0
        @system_grd_rₙ_0 system_control_rₙ = 1
      then
        @system_act_rₙ_0 system_control_rₙ ≔ 2
        @system_act_rₙ_1 components_read_rₙ ≔ COMPONENTS_Rₙ × {0}
        @system_act_rₙ_2 system_connection_i_0_rₙ ≔ <Component_i_Output>
        @system_act_rₙ_3 system_connection_i_j_rₙ ≔ <Component_i_Output>
    end

    convergent event component_0_environment
      any system_control_rₙ_new
      where
        @grd0_0 <Component_0_mode> = 0
        @system_grd_rₙ_0 system_control_rₙ = 2
        @system_grd_rₙ_1 components_read_rₙ[COMPONENTS_Rₙ∖{component_0_rₙ}] = {1} ⇒ system_control_rₙ_new = 0
        @system_grd_rₙ_2 ¬components_read_rₙ[COMPONENTS_Rₙ∖{component_0_rₙ}] = {1} ⇒ system_control_rₙ_new = 2
        @system_grd_rₙ_3 components_read_rₙ(component_0_rₙ) = 0
      then
        @act0_0 <Component_0_mode> ≔ 1
        @act0_1 <Component_0_input> ≔ system_connection_i_0_rₙ
        ... // Update the other inputs of the component 0, if any
        @system_act_rₙ_0 system_control_rₙ ≔ system_control_rₙ_new
        @system_act_rₙ_1 components_read_rₙ(component_0_rₙ) ≔ 1
    end

    convergent event component_j_environment
      any system_control_rₙ_new
      where
        @grd0_0 <Component_j_mode> = 0
        @system_grd_rₙ_0 system_control_rₙ = 2
        @system_grd_rₙ_1 components_read_rₙ[COMPONENTS_Rₙ∖{component_j_rₙ}] = {1} ⇒ system_control_rₙ_new = 0
        @system_grd_rₙ_2 ¬components_read_rₙ[COMPONENTS_Rₙ∖{component_j_rₙ}] = {1} ⇒ system_control_rₙ_new = 2
        @system_grd_rₙ_3 components_read_rₙ(component_j_rₙ) = 0
      then
        @act0_0 <Component_j_mode> ≔ 1
        @act0_1 <Component_j_input> ≔ system_connection_i_j_rₙ
        ... // Update the other inputs of the component j, if any
        @system_act_rₙ_0 system_control_rₙ ≔ system_control_rₙ_new
        @system_act_rₙ_2 components_read_rₙ(component_j_rₙ) ≔ 1
    end
  end
```

Appendix E

The complete model of the general electro-valve after instantiation

```
context GEV_0_Parameters_C0
constants
  GEV_0_diameter_min_val
  GEV_0_diameter_max_val
  GEV_0_CONTROL
  GEV_0_rate
axioms
  @GEV_0_axm0_0 GEV_0_diameter_min_val = 0 // If position of a valve is at minimum, the valve is fully closed (0% open)
  @GEV_0_axm0_1 GEV_0_diameter_max_val = 10 // On contrary, maximum is when the valve is fully open (100% open)
  @GEV_0_axm0_2 GEV_0_CONTROL = {−1,0,1} // -1 - closing, 0 - OFF, 1 - opening
  @GEV_0_axm0_3 GEV_0_rate = GEV_0_diameter_max_val // The rate showing how fast the valve opens
  theorem @GEV_0_axm0_4 GEV_0_rate ≤ GEV_0_diameter_max_val − GEV_0_diameter_min_val
end



machine GEV_0_Behaviour_M0 sees GEV_0_Parameters_C0
variables
  GEV_0_control_I
  GEV_0_flow_I
  GEV_0_flow_O
  GEV_0_mode
  GEV_0_position
invariants
  // Control for the valve: -1 - close, 0 - OFF, 1 - open
  @GEV_0_inv0_0 GEV_0_control_I ∈ GEV_0_CONTROL
  // The flow of fluid coming into the valve
  @GEV_0_inv0_1 GEV_0_flow_I ∈ GEV_0_diameter_min_val..GEV_0_diameter_max_val
  // The flow of fluid coming from the valve
  @GEV_0_inv0_2 GEV_0_flow_O ∈ GEV_0_diameter_min_val..GEV_0_diameter_max_val
  // To obtain a deterministic behaviour of the component, we use an internal variable that specifies the mode
  @GEV_0_inv0_3 GEV_0_mode ∈ 0..1
  // The current state of the plunger in the valve
  @GEV_0_inv0_4 GEV_0_position ∈ GEV_0_diameter_min_val..GEV_0_diameter_max_val
  // The output flow cannot be stronger than the input flow
  @GEV_0_inv0_10 GEV_0_mode = 0 ⇒ GEV_0_flow_O ≤ GEV_0_flow_I
  // The output flow cannot be larger than the opening of the valve
  @GEV_0_inv0_11 GEV_0_flow_O ≤ GEV_0_position
events
  event INITIALISATION // Initially, the valve is closed
    then
      @GEV_0_act0_0 GEV_0_control_I ≔ 0
      @GEV_0_act0_1 GEV_0_flow_I :∈ GEV_0_diameter_min_val..GEV_0_diameter_max_val
      @GEV_0_act0_2 GEV_0_flow_O ≔ GEV_0_diameter_min_val
      @GEV_0_act0_3 GEV_0_mode ≔ 0
      @GEV_0_act0_4 GEV_0_position ≔ GEV_0_diameter_min_val
  end

  event GEV_0_environment // This is the interface with the external world
    where
      @grd0_0 GEV_0_mode = 0
    then
```

```
    @act0_0 GEV_0_mode ≔ 1
    @act0_1 GEV_0_control_I :∈ GEV_0_CONTROL
    @act0_2 GEV_0_flow_I :∈ GEV_0_diameter_min_val‥GEV_0_diameter_max_val
  end

  event GEV_0_opening // While the command is open, the valve should be opening with some rate
   any GEV_0_flow_O_new
   where
    @grd0_0 GEV_0_control_I = 1 // If the command is to open the valve
    @grd0_1 GEV_0_position + GEV_0_rate ≤ GEV_0_diameter_max_val // and it is not completely open
    @grd0_2 GEV_0_mode = 1
     // The valve opens and allows the flow to go through with some rate
    @grd0_3 GEV_0_position + GEV_0_rate < GEV_0_flow_I ⇒ GEV_0_flow_O_new = GEV_0_position + GEV_0_rate
    // but the output flow cannot be stronger than the input one, even if the valve is completely open
    @grd0_4 GEV_0_position + GEV_0_rate ≥ GEV_0_flow_I ⇒ GEV_0_flow_O_new = GEV_0_flow_I
   then
    @act0_0 GEV_0_flow_O ≔ GEV_0_flow_O_new
    @act0_1 GEV_0_mode ≔ 0
    @act0_2 GEV_0_position ≔ GEV_0_position + GEV_0_rate
  end

  event GEV_0_closing // While the command is close, the valve should be closing with some rate
   any GEV_0_flow_O_new
   where
    @grd0_0 GEV_0_control_I = −1 // If the command is to close the valve
    @grd0_1 GEV_0_position − GEV_0_rate ≥ GEV_0_diameter_min_val // and the valve is not completely closed yet
    @grd0_2 GEV_0_mode = 1
    // The valve closes and decreases the flow with some rate
    @grd0_3 GEV_0_position − GEV_0_rate ≤ GEV_0_flow_I ⇒ GEV_0_flow_O_new = GEV_0_position − GEV_0_rate
    @grd0_4 GEV_0_position − GEV_0_rate > GEV_0_flow_I ⇒ GEV_0_flow_O_new = GEV_0_flow_I
    // but if it is open more than the input flow is, the output flow should be updated accordingly
   then
    @act0_0 GEV_0_flow_O ≔ GEV_0_flow_O_new
    @act0_1 GEV_0_mode ≔ 0
    @act0_2 GEV_0_position ≔ GEV_0_position − GEV_0_rate
  end

  // If the command is neither close nor open, or it is not possible to open or close the valve anymore, just stop
  event GEV_0_stop
   any GEV_0_flow_O_new
   where
    @grd0_0 GEV_0_control_I = 0 ∨
        (GEV_0_position − GEV_0_rate < GEV_0_diameter_min_val ∧ GEV_0_control_I = −1) ∨
        (GEV_0_position + GEV_0_rate > GEV_0_diameter_max_val ∧ GEV_0_control_I = 1)
    @grd0_1 GEV_0_mode = 1
    @grd0_2 GEV_0_flow_I < GEV_0_flow_O ⇒ GEV_0_flow_O_new = GEV_0_flow_I
    @grd0_3 GEV_0_flow_I ≥ GEV_0_flow_O ⇒ GEV_0_flow_O_new = GEV_0_flow_O
   then
    @act0_0 GEV_0_mode ≔ 0
    @act0_1 GEV_0_flow_O ≔ GEV_0_flow_O_new
  end
end
```

Appendix F

General electro-valve with a connector

```
context GEV_0_Electrovalves_Connection_C1 extends GEV_0_Parameters_C0
constants SYSTEM_CONTROL_R1
axioms
  @system_axm_r1_0 SYSTEM_CONTROL_R1 = {0,1,2}
end
```

```
machine GEV_0_Electrovalves_Connection_M1 refines GEV_0_Behaviour_M0 sees GEV_0_Electrovalves_Connection_C1
variables
  ... // The variables derived from the GEV model
  system_control_r1
  system_GEV_0_EVs_connection_r1
invariants
  @system_control_r1 system_control_r1 ∈ SYSTEM_CONTROL_R1
  @system_connection_GEV_0_EVs_r1 system_GEV_0_EVs_connection_r1 ∈
                                                   GEV_0_diameter_min_val..GEV_0_diameter_max_val
variant system_control_r1
events
  event INITIALISATION extends INITIALISATION
   then
    @system_control_r1 system_control_r1 ≔ 0
    @system_connection_GEV_0_EVs_r1 system_GEV_0_EVs_connection_r1 ≔ GEV_0_diameter_min_val
  end

  event GEV_0_environment refines GEV_0_environment
   where
    @grd0_0 GEV_0_mode = 0
    @system_grd_r1_0 system_control_r1 = 0
   then
    @act0_0 GEV_0_mode ≔ 1
    @act0_1 GEV_0_control_I :∈ GEV_0_CONTROL
    @act0_2 GEV_0_flow_I :∈ GEV_0_diameter_min_val..GEV_0_diameter_max_val
    @system_act_r1_0 system_control_r1 ≔ 1
  end

  convergent event system_connection_GEV_0_EVs
   where
    @system_grd_r1_0 GEV_0_mode = 0
    @system_grd_r1_1 system_control_r1 = 1
   then
    @system_act_r1_0 system_control_r1 ≔ 0
    @system_act_r1_1 system_GEV_0_EVs_connection_r1 ≔ GEV_0_flow_O
  end
  ...
end
```

Appendix G

General electro-valve connected with the generic component

```
context Electrovalves_Doors_Gears_Generic_C2 extends GEV_0_Electrovalves_Connection_C1
constants SYSTEM_CONTROL_R2
axioms
  @system_axm_r2_0 SYSTEM_CONTROL_R2 = {0,1,2}
end



machine Electrovalves_Doors_Gears_Generic_M2 refines GEV_0_Electrovalves_Connection_M1
sees Electrovalves_Doors_Gears_Generic_C2
variables
  ... // Other variables derived from previous refinements
  GenericComponent_0_I
  GenericComponent_0_O
  GenericComponent_0_mode
  GenericComponent_0_IOrelation
  system_control_r2
invariants
  theorem @GenericComponent_thm0_0 ∀ps,s·ps ∈ ℙ1(ℤ) ∧ finite(ps) ∧ s ∈ ℙ(ps) ∧ card(s) = card(ps) ⇒ s = ps
  @GenericComponent_0_inv0_0 GenericComponent_0_I ∈ ℙ1(ℤ)
  @GenericComponent_0_inv0_1 GenericComponent_0_O ∈ ℙ1(ℤ)
  @GenericComponent_0_inv0_2 GenericComponent_0_mode ∈ 0..1
  @GenericComponent_0_inv0_3 GenericComponent_0_IOrelation ∈ GenericComponent_0_I ↔ GenericComponent_0_O
  @GenericComponent_0_inv0_10 finite(GenericComponent_0_I) ∧ finite(GenericComponent_0_O)
  @GenericComponent_0_inv0_11 dom(GenericComponent_0_IOrelation) = GenericComponent_0_I
  @GenericComponent_0_inv0_12 ran(GenericComponent_0_IOrelation) = GenericComponent_0_O
  @GenericComponent_0_inv0_13 GenericComponent_0_mode = 0 ⇒ GenericComponent_0_O =
                                                     GenericComponent_0_IOrelation[GenericComponent_0_I]
  @system_control_inv_r2_0 system_control_r2 ∈ SYSTEM_CONTROL_R2
  @system_glueinv_r2_1 system_control_r1 = 0 ⇔ system_control_r2 = 0 ∨ system_control_r2 = 2
  @system_glueinv_r2_2 system_control_r1 = 1 ⇔ system_control_r2 = 1
  @system_conn_inv_r2_3 GenericComponent_0_I = GEV_0_diameter_min_val..GEV_0_diameter_max_val
variant system_control_r2
events
  event INITIALISATION
    then
      ... // Initialisation of the variables derived from the previous refinements
      @GenericComponent_0_act0_0 GenericComponent_0_mode ≔ 0
      @GenericComponent_0_act0_1 GenericComponent_0_I, GenericComponent_0_O, GenericComponent_0_IOrelation :∣
                          GenericComponent_0_I' = GEV_0_diameter_min_val..GEV_0_diameter_max_val ∧
                          GenericComponent_0_IOrelation' ∈ GenericComponent_0_I' ↔ GenericComponent_0_O' ∧
                          dom(GenericComponent_0_IOrelation') = GenericComponent_0_I' ∧
                          ran(GenericComponent_0_IOrelation') = GenericComponent_0_O'
      @system_control_r2_0 system_control_r2 ≔ 0
      @system_connection_GEV_0_EVs_r1_1 system_GEV_0_EVs_connection_r1 ≔ GEV_0_diameter_min_val
  end

  event GEV_0_environment refines GEV_0_environment
    where
      @grd0_0 GEV_0_mode = 0
      @system_grd_r1_0 system_control_r2 = 0
    then
```

```
    @act0_0 GEV_0_mode ≔ 1
    @act0_1 GEV_0_control_I :∈ GEV_0_CONTROL
    @act0_2 GEV_0_flow_I :∈ GEV_0_diameter_min_val..GEV_0_diameter_max_val
    @system_act_r1_0 system_control_r2 ≔ 1
  end

  event system_connection_GEV_0_EVs refines system_connection_GEV_0_EVs
   where
    @system_grd_r1_0 GEV_0_mode = 0
    @system_grd_r2_0 system_control_r2 = 1
   then
    @system_act_r1_0 system_GEV_0_EVs_connection_r1 ≔ GEV_0_flow_O
    @system_act_r2_0 system_control_r2 ≔ 2
  end

  convergent event GenericComponent_0_environment
   where
    @grd0_0 GenericComponent_0_mode = 0
    @system_grd_r2_0 system_control_r2 = 2
   then
    @act0_0 GenericComponent_0_mode ≔ 1
    @act0_1 GenericComponent_0_I ≔ GEV_0_diameter_min_val..GEV_0_diameter_max_val //
{system_GEV_0_EVs_connection_r1}
    @system_act_r2_0 system_control_r2 ≔ 0
  end
  …
end
```

Appendix H

General electro-valve connected with the specific electro-valves that refine the generic component

```
context Electrovalves_Doors_Gears_C3 extends Electrovalves_Doors_Gears_Generic_C2
sets
 COMPONENTS_R3
constants
 evalve_0_r3 evalve_1_r3 evalve_2_r3 evalve_3_r3
 evalve_0_diameter_min_val
 evalve_0_diameter_max_val
 evalve_0_CONTROL
 evalve_0_rate
 ... // Parameters of other valves
axioms
 // If position of a valve is at minimum, the valve is fully closed (0% open)
 @evalve_0_axm_0 evalve_0_diameter_min_val = 0
 // On contrary, maximum means that the valve is fully open (100% open)
 @evalve_0_axm_1 evalve_0_diameter_max_val = 10
 @evalve_0_axm_2 evalve_0_CONTROL = {−1,0,1} // -1 - closing, 0 - OFF, 1 - opening
 @evalve_0_axm_3 evalve_0_rate = evalve_0_diameter_max_val // The rate showing how fast the valve opens
 // The rate showing how fast the valve opens or closes. If it equals 100, the valve is simply open/close.
 theorem @evalve_0_axm_4 evalve_0_rate ≤ evalve_0_diameter_max_val − evalve_0_diameter_min_val
 @evalve_1_axm_0 evalve_1_diameter_min_val = 0
 @evalve_1_axm_1 evalve_1_diameter_max_val = 10
 @evalve_1_axm_2 evalve_1_CONTROL = {−1,0,1}
 @evalve_1_axm_3 evalve_1_rate = evalve_1_diameter_max_val
 theorem @evalve_1_axm_4 evalve_1_rate ≤ evalve_1_diameter_max_val − evalve_1_diameter_min_val
 @evalve_2_axm_0 evalve_2_diameter_min_val = 0
 @evalve_2_axm_1 evalve_2_diameter_max_val = 10
 @evalve_2_axm_2 evalve_2_CONTROL = {−1,0,1}
 @evalve_2_axm_3 evalve_2_rate = evalve_2_diameter_max_val
 theorem @evalve_2_axm_4 evalve_2_rate ≤ evalve_2_diameter_max_val − evalve_2_diameter_min_val
 @evalve_3_axm_0 evalve_3_diameter_min_val = 0
 @evalve_3_axm_1 evalve_3_diameter_max_val = 10
 @evalve_3_axm_2 evalve_3_CONTROL = {−1,0,1}
 @evalve_3_axm_3 evalve_3_rate = evalve_3_diameter_max_val
 theorem @evalve_3_axm_4 evalve_3_rate ≤ evalve_3_diameter_max_val − evalve_3_diameter_min_val
 // Theorems to support the connectivity between the valves
 theorem @system_thm_r3_0 evalve_0_diameter_max_val = GEV_0_diameter_max_val
 theorem @system_thm_r3_1 evalve_1_diameter_max_val = GEV_0_diameter_max_val
 theorem @system_thm_r3_2 evalve_2_diameter_max_val = GEV_0_diameter_max_val
 theorem @system_thm_r3_3 evalve_3_diameter_max_val = GEV_0_diameter_max_val
 // Definition of the components (i.e., electro-valves) to be introduced
 @system_evalve_set partition(COMPONENTS_R3, {evalve_0_r3}, {evalve_1_r3}, {evalve_2_r3}, {eevalve_3_r3})
end
```

```
machine M3_Electrovalves_Doors_Gears refines M2_Electrovalves_Doors_Gears_Generic
sees Electrovalves_Doors_Gears_C3
variables
   GEV_0_control_I            evalve_1_flow_I            evalve_3_flow_I
   GEV_0_flow_I               evalve_1_flow_O            evalve_3_flow_O
```

```
    GEV_0_flow_O              evalve_1_mode             evalve_3_mode
    GEV_0_mode                evalve_1_position         evalve_3_position
    GEV_0_position            evalve_2_control_I        system_GEV_0_EVs_connection_r3_0
    evalve_0_control_I        evalve_2_flow_I           system_GEV_0_EVs_connection_r3_1
    evalve_0_flow_I           evalve_2_flow_O           system_GEV_0_EVs_connection_r3_2
    evalve_0_flow_O           evalve_2_mode             system_GEV_0_EVs_connection_r3_3
    evalve_0_mode             evalve_2_position         GenericComponent_0_mode
    evalve_0_position         evalve_3_control_I        system_control_r2
    evalve_1_control_I                                  valves_read_r3
invariants
  // Control for the valve: -1 - close, 0 - OFF, 1 - open
  @evalve_0_inv0_0 evalve_0_control_I ∈ evalve_0_CONTROL
  // The flow of fluid coming into the valve
  @evalve_0_inv0_1 evalve_0_flow_I ∈ evalve_0_diameter_min_val..evalve_0_diameter_max_val
  // The flow of fluid coming from the valve
  @evalve_0_inv0_2 evalve_0_flow_O ∈ evalve_0_diameter_min_val..evalve_0_diameter_max_val
  // To obtain a deterministic behaviour of the component, we use an internal variable that specifies the mode
  @evalve_0_inv0_3 evalve_0_mode ∈ 0..1
  // The current state of the valve
  @evalve_0_inv0_4 evalve_0_position ∈ evalve_0_diameter_min_val..evalve_0_diameter_max_val
  // The output flow cannot be stronger than the input flow
  @evalve_0_inv0_10 evalve_0_mode = 0 ⇒ evalve_0_flow_O ≤ evalve_0_flow_I
  // The output flow cannot be larger than the opening of the valve
  @evalve_0_inv0_11 evalve_0_flow_O ≤ evalve_0_position
  @evalve_1_inv0_0 evalve_1_control_I ∈ evalve_1_CONTROL
  @evalve_1_inv0_1 evalve_1_flow_I ∈ evalve_1_diameter_min_val..evalve_1_diameter_max_val
  @evalve_1_inv0_2 evalve_1_flow_O ∈ evalve_1_diameter_min_val..evalve_1_diameter_max_val
  @evalve_1_inv0_3 evalve_1_mode ∈ 0..1
  @evalve_1_inv0_4 evalve_1_position ∈ evalve_1_diameter_min_val..evalve_1_diameter_max_val
  @evalve_1_inv0_10 evalve_1_mode = 0 ⇒ evalve_1_flow_O ≤ evalve_1_flow_I
  @evalve_1_inv0_11 evalve_1_flow_O ≤ evalve_1_position
  @evalve_2_inv0_0 evalve_2_control_I ∈ evalve_2_CONTROL
  @evalve_2_inv0_1 evalve_2_flow_I ∈ evalve_2_diameter_min_val..evalve_2_diameter_max_val
  @evalve_2_inv0_2 evalve_2_flow_O ∈ evalve_2_diameter_min_val..evalve_2_diameter_max_val
  @evalve_2_inv0_3 evalve_2_mode ∈ 0..1
  @evalve_2_inv0_4 evalve_2_position ∈ evalve_2_diameter_min_val..evalve_2_diameter_max_val
  @evalve_2_inv0_10 evalve_2_mode = 0 ⇒ evalve_2_flow_O ≤ evalve_2_flow_I
  @evalve_2_inv0_11 evalve_2_flow_O ≤ evalve_2_position
  @evalve_3_inv0_0 evalve_3_control_I ∈ evalve_3_CONTROL
  @evalve_3_inv0_1 evalve_3_flow_I ∈ evalve_3_diameter_min_val..evalve_3_diameter_max_val
  @evalve_3_inv0_2 evalve_3_flow_O ∈ evalve_3_diameter_min_val..evalve_3_diameter_max_val
  @evalve_3_inv0_3 evalve_3_mode ∈ 0..1
  @evalve_3_inv0_4 evalve_3_position ∈ evalve_3_diameter_min_val..evalve_3_diameter_max_val
  @evalve_3_inv0_10 evalve_3_mode = 0 ⇒ evalve_3_flow_O ≤ evalve_3_flow_I
  @evalve_3_inv0_11 evalve_3_flow_O ≤ evalve_3_position
  // Gluing invariants
  @system_glueinv_r3_0 GenericComponent_0_I ⊆
                 evalve_0_diameter_min_val..evalve_0_diameter_max_val ∪
                 evalve_1_diameter_min_val..evalve_1_diameter_max_val ∪
                 evalve_2_diameter_min_val..evalve_2_diameter_max_val ∪
                 evalve_3_diameter_min_val..evalve_3_diameter_max_val
  @system_glueinv_r3_1 GenericComponent_0_O ⊆
                 evalve_0_diameter_min_val..evalve_0_diameter_max_val ∪
                 evalve_1_diameter_min_val..evalve_1_diameter_max_val ∪
                 evalve_2_diameter_min_val..evalve_2_diameter_max_val ∪
                 evalve_3_diameter_min_val..evalve_3_diameter_max_val
  @system_glueinv_r3_2 GenericComponent_0_IOrelation ⊆
                 (evalve_0_diameter_min_val..evalve_0_diameter_max_val ∪
```

```
                evalve_1_diameter_min_val..evalve_1_diameter_max_val ∪
                evalve_2_diameter_min_val..evalve_2_diameter_max_val ∪
                evalve_3_diameter_min_val..evalve_3_diameter_max_val) ×
                (evalve_0_diameter_min_val..evalve_0_diameter_max_val ∪
                evalve_1_diameter_min_val..evalve_1_diameter_max_val ∪
                evalve_2_diameter_min_val..evalve_2_diameter_max_val ∪
                evalve_3_diameter_min_val..evalve_3_diameter_max_val)
 @system_glueinv_r3_3 system_GEV_0_EVs_connection_r1 = system_GEV_0_EVs_connection_r3_0 ∧
                system_GEV_0_EVs_connection_r1 = system_GEV_0_EVs_connection_r3_1 ∧
                system_GEV_0_EVs_connection_r1 = system_GEV_0_EVs_connection_r3_2 ∧
                system_GEV_0_EVs_connection_r1 = system_GEV_0_EVs_connection_r3_3
 // Types of connectors
 @system_connection_GEV_0_EVs_r3_4 system_GEV_0_EVs_connection_r3_0 ∈
                                            evalve_0_diameter_min_val..evalve_0_diameter_max_val
 @system_connection_GEV_0_EVs_r3_5 system_GEV_0_EVs_connection_r3_1 ∈
                                            evalve_1_diameter_min_val..evalve_1_diameter_max_val
 @system_connection_GEV_0_EVs_r3_6 system_GEV_0_EVs_connection_r3_2 ∈
                                            evalve_2_diameter_min_val..evalve_2_diameter_max_val
 @system_connection_GEV_0_EVs_r3_7 system_GEV_0_EVs_connection_r3_3 ∈
                                            evalve_3_diameter_min_val..evalve_3_diameter_max_val
 // Restricting the number of reads per iteration
 @system_valves_worked_r3_8 valves_read_r3 ∈ COMPONENTS_R3 → 0..1
 // "Gluing" the old and the new data
 @system_inv_r3_9 system_control_r2 = 0 ∧ GenericComponent_0_mode = 1 ⇒ valves_read_r3[COMPONENTS_R3] = {0}

variant card(COMPONENTS_R3) − valves_read_r3(evalve_0_r3) − valves_read_r3(evalve_1_r3) −
                                            valves_read_r3(evalve_2_r3) − valves_read_r3(eevalve_3_r3)

events
 event INITIALISATION
  with
   @GenericComponent_0_I' GenericComponent_0_I' = evalve_0_diameter_min_val..evalve_0_diameter_max_val ∪
                evalve_1_diameter_min_val..evalve_1_diameter_max_val ∪
                evalve_2_diameter_min_val..evalve_2_diameter_max_val ∪
                evalve_3_diameter_min_val..evalve_3_diameter_max_val
   @GenericComponent_0_O' GenericComponent_0_O' = evalve_0_diameter_min_val..evalve_0_diameter_max_val ∪
                evalve_1_diameter_min_val..evalve_1_diameter_max_val ∪
                evalve_2_diameter_min_val..evalve_2_diameter_max_val ∪
                evalve_3_diameter_min_val..evalve_3_diameter_max_val
   @GenericComponent_0_IOrelation' GenericComponent_0_IOrelation' =
                    (evalve_0_diameter_min_val..evalve_0_diameter_max_val ∪
                    evalve_1_diameter_min_val..evalve_1_diameter_max_val ∪
                    evalve_2_diameter_min_val..evalve_2_diameter_max_val ∪
                    evalve_3_diameter_min_val..evalve_3_diameter_max_val)×
                    (evalve_0_diameter_min_val..evalve_0_diameter_max_val ∪
                    evalve_1_diameter_min_val..evalve_1_diameter_max_val ∪
                    evalve_2_diameter_min_val..evalve_2_diameter_max_val ∪
                    evalve_3_diameter_min_val..evalve_3_diameter_max_val)
  then
   @GEV_0_act0_0 GEV_0_control_I ≔ 0
   @GEV_0_act0_1 GEV_0_flow_I :∈ GEV_0_diameter_min_val..GEV_0_diameter_max_val
   @GEV_0_act0_2 GEV_0_flow_O ≔ GEV_0_diameter_min_val
   @GEV_0_act0_3 GEV_0_mode ≔ 0
   @GEV_0_act0_4 GEV_0_position ≔ GEV_0_diameter_min_val
   @system_control_r2_0 system_control_r2 ≔ 0
   @GenericComponent_0_act0_0 GenericComponent_0_mode ≔ 0
   @evalve_0_act0_0 evalve_0_control_I ≔ 0
   @evalve_0_act0_1 evalve_0_flow_I :∈ evalve_0_diameter_min_val..evalve_0_diameter_max_val
   @evalve_0_act0_2 evalve_0_flow_O ≔ evalve_0_diameter_min_val
```

```
    @evalve_0_act0_3 evalve_0_mode ≔ 0
    @evalve_0_act0_4 evalve_0_position ≔ evalve_0_diameter_min_val
    @evalve_1_act0_0 evalve_1_control_I ≔ 0
    @evalve_1_act0_1 evalve_1_flow_I :∈ evalve_1_diameter_min_val..evalve_1_diameter_max_val
    @evalve_1_act0_2 evalve_1_flow_O ≔ evalve_1_diameter_min_val
    @evalve_1_act0_3 evalve_1_mode ≔ 0
    @evalve_1_act0_4 evalve_1_position ≔ evalve_1_diameter_min_val
    @evalve_2_act0_0 evalve_2_control_I ≔ 0
    @evalve_2_act0_1 evalve_2_flow_I :∈ evalve_2_diameter_min_val..evalve_2_diameter_max_val
    @evalve_2_act0_2 evalve_2_flow_O ≔ evalve_2_diameter_min_val
    @evalve_2_act0_3 evalve_2_mode ≔ 0
    @evalve_2_act0_4 evalve_2_position ≔ evalve_2_diameter_min_val
    @evalve_3_act0_0 evalve_3_control_I ≔ 0
    @evalve_3_act0_1 evalve_3_flow_I :∈ evalve_3_diameter_min_val..evalve_3_diameter_max_val
    @evalve_3_act0_2 evalve_3_flow_O ≔ evalve_3_diameter_min_val
    @evalve_3_act0_3 evalve_3_mode ≔ 0
    @evalve_3_act0_4 evalve_3_position ≔ evalve_3_diameter_min_val
    @system_act_r3_0 system_GEV_0_EVs_connection_r3_0 ≔ evalve_0_diameter_min_val
    @system_act_r3_1 system_GEV_0_EVs_connection_r3_1 ≔ evalve_1_diameter_min_val
    @system_act_r3_2 system_GEV_0_EVs_connection_r3_2 ≔ evalve_2_diameter_min_val
    @system_act_r3_3 system_GEV_0_EVs_connection_r3_3 ≔ evalve_3_diameter_min_val
    @system_valves_worked_r3_4 valves_read_r3 ≔ COMPONENTS_R3 × {0}
end

event GEV_0_environment extends GEV_0_environment
end

event system_connection_GEV_0_EVs refines system_connection_GEV_0_EVs
  where
    @system_grd_r1_0 GEV_0_mode = 0
    @system_grd_r3_0 system_control_r2 = 1
  then
    @system_act_r3_0 system_GEV_0_EVs_connection_r3_0 ≔ GEV_0_flow_O
    @system_act_r3_1 system_GEV_0_EVs_connection_r3_1 ≔ GEV_0_flow_O
    @system_act_r3_2 system_GEV_0_EVs_connection_r3_2 ≔ GEV_0_flow_O
    @system_act_r3_3 system_GEV_0_EVs_connection_r3_3 ≔ GEV_0_flow_O
    @system_act_r3_4 system_control_r2 ≔ 2
end

convergent event evalve_0_environment
  where
    @grd0_0 evalve_0_mode = 0
    @system_grd_r3_0 system_control_r2 = 2
    @system_evalve_0_worked_grd valves_read_r3(evalve_0_r3) = 0
  then
    @act0_0 evalve_0_mode ≔ 1
    @act0_1 evalve_0_control_I :∈ evalve_0_CONTROL
    @act0_2 evalve_0_flow_I ≔ system_GEV_0_EVs_connection_r3_0
    @system_evalve_0_worked_act valves_read_r3(evalve_0_r3) ≔ 1
end

convergent event evalve_1_environment
  where
    @grd0_0 evalve_1_mode = 0
    @system_grd_r3_0 system_control_r2 = 2
    @system_evalve_1_worked_grd valves_read_r3(evalve_1_r3) = 0
  then
    @act0_0 evalve_1_mode ≔ 1
```

```
    @act0_1 evalve_1_control_I :∈ evalve_1_CONTROL
    @act0_2 evalve_1_flow_I ≔ system_GEV_0_EVs_connection_r3_1
    @system_evalve_1_worked_act valves_read_r3(evalve_1_r3) ≔ 1
  end

  convergent event evalve_2_environment
   where
    @grd0_0 evalve_2_mode = 0
    @system_grd_r3_0 system_control_r2 = 2
    @system_evalve_0_worked_grd valves_read_r3(evalve_2_r3) = 0
   then
    @act0_0 evalve_2_mode ≔ 1
    @act0_1 evalve_2_control_I :∈ evalve_2_CONTROL
    @act0_2 evalve_2_flow_I ≔ system_GEV_0_EVs_connection_r3_2
    @system_evalve_1_worked_act valves_read_r3(evalve_2_r3) ≔ 1
  end

  convergent event evalve_3_environment
   where
    @grd0_0 evalve_3_mode = 0
    @system_grd_r3_0 system_control_r2 = 2
    @system_evalve_0_worked_grd valves_read_r3(evalve_3_r3) = 0
   then
    @act0_0 evalve_3_mode ≔ 1
    @act0_1 evalve_3_control_I :∈ evalve_3_CONTROL
    @act0_2 evalve_3_flow_I ≔ system_GEV_0_EVs_connection_r3_3
    @system_evalve_1_worked_act valves_read_r3(evalve_3_r3) ≔ 1
  end

  event GenericComponent_0_environment refines GenericComponent_0_environment
   where
    @grd0_0 GenericComponent_0_mode = 0
    @system_grd_r2_0 system_control_r2 = 2
    @system_grd_r3_0 valves_read_r3[COMPONENTS_R3] = {1}
   then
    @act0_0 GenericComponent_0_mode ≔ 1
    @system_act_r2_0 system_control_r2 ≔ 0
    @system_act_r3_0 valves_read_r3 ≔ COMPONENTS_R3 × {0}
  end
  …
end
```

Appendix I

General-electro valve connected to the doors/gears electro-valves with connections for the cylinders of doors

```
context EVs_Doors_Connection_C4 extends Electrovalves_Doors_Gears_C3
constants
  SYSTEM_CONTROL_R4_CAP
  SYSTEM_CONTROL_R4_HEAD
axioms
  @system_control_axm_r4_0 SYSTEM_CONTROL_R4_CAP = {0,1,2}
  @system_control_axm_r4_1 SYSTEM_CONTROL_R4_HEAD = {0,1,2}
end



machine EVs_Doors_Connection_M4 refines Electrovalves_Doors_Gears_M3 sees EVs_Doors_Connection_C4
variables
    GEV_0_control_I            valve_1_flow_O         valve_3_position
    GEV_0_flow_I               valve_1_mode           system_GEV_0_EVs_connection_r3_0
    GEV_0_flow_O               valve_1_position       system_GEV_0_EVs_connection_r3_1
    GEV_0_mode                 valve_2_control_I      system_GEV_0_EVs_connection_r3_2
    GEV_0_position             valve_2_flow_I         system_GEV_0_EVs_connection_r3_3
    valve_0_control_I          valve_2_flow_O         GenericComponent_0_mode
    valve_0_flow_I             valve_2_mode           system_control_r2
    valve_0_flow_O             valve_2_position       system_connection_EVs_Doors_r4_cap
    valve_0_mode               valve_3_control_I      system_connection_EVs_Doors_r4_head
    valve_0_position           valve_3_flow_I         system_control_r4_cap
    valve_1_control_I          valve_3_flow_O         system_control_r4_head
    valve_1_flow_I             valve_3_mode           valves_read_r3
invariants
  // Control variables
  @system_control_inv_r4_0 system_control_r4_cap ∊ SYSTEM_CONTROL_R4_CAP
  @system_control_inv_r4_1 system_control_r4_head ∊ SYSTEM_CONTROL_R4_HEAD
  // Connection variables
  @system_connection_inv_EVs_Doors_r4_2 system_connection_EVs_Doors_r4_cap ∊
                                               valve_0_diameter_min_val..valve_0_diameter_max_val
  @system_connection_inv_EVs_Doors_r4_3 system_connection_EVs_Doors_r4_head ∊
                                               valve_1_diameter_min_val..valve_1_diameter_max_val
variant system_control_r4_cap + system_control_r4_head

events
  event INITIALISATION extends INITIALISATION
    then
      @system_act_r4_0 system_control_r4_cap ≔ 0
      @system_act_r4_1 system_control_r4_head ≔ 0
      @system_connection_act_r4_2 system_connection_EVs_Doors_r4_cap ≔ valve_0_diameter_min_val
      @system_connection_act_r4_3 system_connection_EVs_Doors_r4_head ≔ valve_1_diameter_min_val
  end

  event GEV_0_environment extends GEV_0_environment
  end

  event system_connection_GEV_0_EVs extends system_connection_GEV_0_EVs
  end
```

```
  event valve_0_environment extends valve_0_environment
    where
      @system_control_grd_r4_0 system_control_r4_cap = 0
    then
      @system_control_act_r4_0 system_control_r4_cap ≔ 1
  end

  event valve_1_environment extends valve_1_environment
    where
      @system_control_grd_r4_0 system_control_r4_head = 0
    then
      @system_control_act_r4_0 system_control_r4_head ≔ 1
  end

  event valve_2_environment extends valve_2_environment
  end

  event valve_3_environment extends valve_3_environment
  end

  event GenericComponent_0_environement extends GenericComponent_0_environement
  end

  convergent event system_connection_EVs_Doors_cap
    where
      @system_grd_r4_0 valve_0_mode = 0
      @system_grd_r4_1 system_control_r4_cap = 1
    then
      @system_act_r4_0 system_control_r4_cap ≔ 0
      @system_connection_act_r4_1 system_connection_EVs_Doors_r4_cap ≔ valve_0_flow_O
  end

  convergent event system_connection_EVs_Doors_head
    where
      @system_grd_r4_0 valve_1_mode = 0
      @system_grd_r4_1 system_control_r4_head = 1
    then
      @system_act_r4_0 system_control_r4_head ≔ 0
      @system_connection_act_r4_1 system_connection_EVs_Doors_r4_head ≔ valve_1_flow_O
  end
…
end
```

Appendix J

General-electro valve connected to the doors/gears electro-valves with connections for and the cylinders for doors

```
context Cylinders_Doors_C5 extends EVs_Doors_Connection_C4
sets COMPONENTS_R5
constants
   cylinder_0_input_diameter_min_val      cylinder_1_cap_pos                     SYSTEM_CONTROL_R5_CAP
   cylinder_0_input_diameter_max_val      cylinder_1_head_pos                    SYSTEM_CONTROL_R5_HEAD
   cylinder_0_cap_pos                     cylinder_2_input_diameter_min_val      cylinder_0_r5
   cylinder_0_head_pos                    cylinder_2_input_diameter_max_val      cylinder_1_r5
   cylinder_1_input_diameter_min_val      cylinder_2_cap_pos                     cylinder_2_r5
   cylinder_1_input_diameter_max_val      cylinder_2_head_pos
axioms
 // 0 stands for no liquid flowing into the cylinder (0% open)
 @cylinder_0_axm_0 cylinder_0_input_diameter_min_val = 0
 // 100 stands for maximum velocity the piston can move inside the cylinder (100% open)
 @cylinder_0_axm_1 cylinder_0_input_diameter_max_val = 10
 @cylinder_0_axm_2 cylinder_0_cap_pos = 0
 // We do not provide any value to define the length of the cylinder, but it has to be done according to the rules
 @cylinder_0_axm_3 cylinder_0_head_pos ∈ ℕ1
 @cylinder_1_axm_0 cylinder_1_input_diameter_min_val = 0
 @cylinder_1_axm_1 cylinder_1_input_diameter_max_val = 10
 @cylinder_1_axm_2 cylinder_1_cap_pos = 0
 @cylinder_1_axm_3 cylinder_1_head_pos ∈ ℕ1
 @cylinder_2_axm_0 cylinder_2_input_diameter_min_val = 0
 @cylinder_2_axm_1 cylinder_2_input_diameter_max_val = 10
 @cylinder_2_axm_2 cylinder_2_cap_pos = 0
 @cylinder_2_axm_3 cylinder_2_head_pos ∈ ℕ1
 // Theorems to support connection conditions
 theorem @system_axm_r5_0 cylinder_0_input_diameter_max_val = valve_0_diameter_max_val
 theorem @system_axm_r5_1 cylinder_1_input_diameter_max_val = valve_0_diameter_max_val
 theorem @system_axm_r5_2 cylinder_2_input_diameter_max_val = valve_0_diameter_max_val
 // Control values
 @system_axm_r5_3 SYSTEM_CONTROL_R5_CAP = {0,1,2}
 @system_axm_r5_4 SYSTEM_CONTROL_R5_HEAD = {0,1,2}
 // Definition of the cylinders to be introduced
 @system_axm_r5_5 partition(COMPONENTS_R5, {cylinder_0_r5}, {cylinder_1_r5}, {cylinder_2_r5})
end
```

```
machine Cylinders_Doors_M5 refines EVs_Doors_Connection_M4 sees Cylinders_Doors_C5
variables
    GEV_0_control_I                         system_GEV_0_EVs_connection_r3_2
    GEV_0_flow_I                            system_GEV_0_EVs_connection_r3_3
    GEV_0_flow_O                            GenericComponent_0_mode
    GEV_0_mode                              system_control_r2
    GEV_0_position                          valves_read_r3
    valve_0_control_I                       cylinder_0_piston_position_O
    valve_0_flow_I                          cylinder_0_flow_cap_I
    valve_0_flow_O                          cylinder_0_flow_head_I
    valve_0_mode                            cylinder_0_mode
  valve_0_position                          cylinder_1_piston_position_O
```

```
  valve_1_control_I
  valve_1_flow_I
  valve_1_flow_O
  valve_1_mode
  valve_1_position
  valve_2_control_I
  valve_2_flow_I
  valve_2_flow_O
   valve_2_mode
   valve_2_position
   valve_3_control_I
   valve_3_flow_I
   valve_3_flow_O
   valve_3_mode
   valve_3_position
   system_GEV_0_EVs_connection_r3_0
   system_GEV_0_EVs_connection_r3_1
  cylinder_1_flow_cap_I
  cylinder_1_flow_head_I
  cylinder_1_mode
  cylinder_2_piston_position_O
  cylinder_2_flow_cap_I
  cylinder_2_flow_head_I
  cylinder_2_mode
  system_connection_EVs_Doors_r6_cap_0
  system_connection_EVs_Doors_r6_cap_1
  system_connection_EVs_Doors_r6_cap_2
  system_connection_EVs_Doors_r6_head_0
  system_connection_EVs_Doors_r6_head_1
  system_connection_EVs_Doors_r6_head_2
  system_control_r5_cap
  system_control_r5_head
  cylinders_read_r5
invariants
  // Current position of the piston in the cylinder
  @cylinder_0_inv0_0 cylinder_0_piston_position_O ∈ cylinder_0_cap_pos..cylinder_0_head_pos
  // Input to move the piston to the right
  @cylinder_0_inv0_1 cylinder_0_flow_cap_I ∈ cylinder_0_input_diameter_min_val..cylinder_0_input_diameter_max_val
  // Input to move the piston to the left
  @cylinder_0_inv0_2 cylinder_0_flow_head_I ∈ cylinder_0_input_diameter_min_val..cylinder_0_input_diameter_max_val
  @cylinder_0_inv0_3 cylinder_0_mode ∈ 0..1
  @cylinder_1_inv0_0 cylinder_1_piston_position_O ∈ cylinder_1_cap_pos..cylinder_1_head_pos
  @cylinder_1_inv0_1 cylinder_1_flow_cap_I ∈ cylinder_1_input_diameter_min_val..cylinder_1_input_diameter_max_val
  @cylinder_1_inv0_2 cylinder_1_flow_head_I ∈ cylinder_1_input_diameter_min_val..cylinder_1_input_diameter_max_val
  @cylinder_1_inv0_3 cylinder_1_mode ∈ 0..1
  @cylinder_2_inv0_0 cylinder_2_piston_position_O ∈ cylinder_2_cap_pos..cylinder_2_head_pos
  @cylinder_2_inv0_1 cylinder_2_flow_cap_I ∈ cylinder_2_input_diameter_min_val..cylinder_2_input_diameter_max_val
  @cylinder_2_inv0_2 cylinder_2_flow_head_I ∈ cylinder_2_input_diameter_min_val..cylinder_2_input_diameter_max_val
  @cylinder_2_inv0_3 cylinder_2_mode ∈ 0..1
  // Connectors
  @system_connection_EVs_Doors_r5_cap_0 system_connection_EVs_Doors_r6_cap_0 ∈
                cylinder_0_input_diameter_min_val..cylinder_0_input_diameter_max_val
  @system_connection_EVs_Doors_r5_cap_1 system_connection_EVs_Doors_r6_cap_1 ∈
                cylinder_1_input_diameter_min_val..cylinder_1_input_diameter_max_val
  @system_connection_EVs_Doors_r5_cap_2 system_connection_EVs_Doors_r6_cap_2 ∈
                cylinder_2_input_diameter_min_val..cylinder_2_input_diameter_max_val
  @system_connection_EVs_Doors_r5_head_3 system_connection_EVs_Doors_r6_head_0 ∈
                cylinder_0_input_diameter_min_val..cylinder_0_input_diameter_max_val
  @system_connection_EVs_Doors_r5_head_4 system_connection_EVs_Doors_r6_head_1 ∈
                cylinder_1_input_diameter_min_val..cylinder_1_input_diameter_max_val
  @system_connection_EVs_Doors_r5_head_5 system_connection_EVs_Doors_r6_head_2 ∈
                cylinder_2_input_diameter_min_val..cylinder_2_input_diameter_max_val
  // Gluing invariants
  @system_glueinv_r5_6 system_connection_EVs_Doors_r4_cap = system_connection_EVs_Doors_r6_cap_0 ∧
        system_connection_EVs_Doors_r4_cap = system_connection_EVs_Doors_r6_cap_1 ∧
        system_connection_EVs_Doors_r4_cap = system_connection_EVs_Doors_r6_cap_2
  @system_glueinv_r5_7 system_connection_EVs_Doors_r4_head = system_connection_EVs_Doors_r6_head_0 ∧
        system_connection_EVs_Doors_r4_head = system_connection_EVs_Doors_r6_head_1 ∧
        system_connection_EVs_Doors_r4_head = system_connection_EVs_Doors_r6_head_2
  @system_glueinv_r5_8 system_control_r4_cap = 0 ⇔ system_control_r5_cap = 0 ∨ system_control_r5_cap = 2
  @system_glueinv_r5_9 system_control_r4_cap = 1 ⇔ system_control_r5_cap = 1
  @system_glueinv_r5_10 system_control_r4_head = 0 ⇔ system_control_r5_head = 0 ∨ system_control_r5_head = 2
  @system_glueinv_r5_11 system_control_r4_head = 1 ⇔ system_control_r5_head = 1
  @system_cylinders_read_r5_12 cylinders_read_r5 ∈ COMPONENTS_R5 → 0..1
```

```
variant card(COMPONENTS_R5) − cylinders_read_r5(cylinder_0_r5) − cylinders_read_r5(cylinder_1_r5) −
                                                              cylinders_read_r5(cylinder_2_r5)
events
 event INITIALISATION   then
   @GEV_0_act0_0 GEV_0_control_I ≔ 0
   @GEV_0_act0_1 GEV_0_flow_I :∈ GEV_0_diameter_min_val..GEV_0_diameter_max_val
   @GEV_0_act0_2 GEV_0_flow_O ≔ GEV_0_diameter_min_val
   @GEV_0_act0_3 GEV_0_mode ≔ 0
   @GEV_0_act0_4 GEV_0_position ≔ GEV_0_diameter_min_val
   @GenericComponent_act0_0 GenericComponent_0_mode ≔ 0
   @system_act_r2_0 system_control_r2 ≔ 0
   @valve_0_act0_0 valve_0_control_I ≔ 0
   @valve_0_act0_1 valve_0_flow_I :∈ valve_0_diameter_min_val..valve_0_diameter_max_val
   @valve_0_act0_2 valve_0_flow_O ≔ valve_0_diameter_min_val
   @valve_0_act0_3 valve_0_mode ≔ 0
   @valve_0_act0_4 valve_0_position ≔ valve_0_diameter_min_val
   @valve_1_act0_0 valve_1_control_I ≔ 0
   @valve_1_act0_1 valve_1_flow_I :∈ valve_1_diameter_min_val..valve_1_diameter_max_val
   @valve_1_act0_2 valve_1_flow_O ≔ valve_1_diameter_min_val
   @valve_1_act0_3 valve_1_mode ≔ 0
   @valve_1_act0_4 valve_1_position ≔ valve_1_diameter_min_val
   @valve_2_act0_0 valve_2_control_I ≔ 0
   @valve_2_act0_1 valve_2_flow_I :∈ valve_2_diameter_min_val..valve_2_diameter_max_val
   @valve_2_act0_2 valve_2_flow_O ≔ valve_2_diameter_min_val
   @valve_2_act0_3 valve_2_mode ≔ 0
   @valve_2_act0_4 valve_2_position ≔ valve_2_diameter_min_val
   @valve_3_act0_0 valve_3_control_I ≔ 0
   @valve_3_act0_1 valve_3_flow_I :∈ valve_3_diameter_min_val..valve_3_diameter_max_val
   @valve_3_act0_2 valve_3_flow_O ≔ valve_3_diameter_min_val
   @valve_3_act0_3 valve_3_mode ≔ 0
   @valve_3_act0_4 valve_3_position ≔ valve_3_diameter_min_val
   @system_act_r3_0 system_GEV_0_EVs_connection_r3_0 ≔ valve_0_diameter_min_val
   @system_act_r3_1 system_GEV_0_EVs_connection_r3_1 ≔ valve_1_diameter_min_val
   @system_act_r3_2 system_GEV_0_EVs_connection_r3_2 ≔ valve_2_diameter_min_val
   @system_act_r3_3 system_GEV_0_EVs_connection_r3_3 ≔ valve_3_diameter_min_val
   @system_valves_worked_r3_4 valves_read_r3 ≔ EVALVES × {0}
   @cylinder_0_act0_0 cylinder_0_piston_position_O :∈ cylinder_0_cap_pos..cylinder_0_head_pos
   @cylinder_0_act0_1 cylinder_0_flow_cap_I ≔ cylinder_0_input_diameter_min_val
   @cylinder_0_act0_2 cylinder_0_flow_head_I ≔ cylinder_0_input_diameter_min_val
   @cylinder_0_act0_3 cylinder_0_mode ≔ 0
   @cylinder_1_act0_0 cylinder_1_piston_position_O :∈ cylinder_1_cap_pos..cylinder_1_head_pos
   @cylinder_1_act0_1 cylinder_1_flow_cap_I ≔ cylinder_1_input_diameter_min_val
   @cylinder_1_act0_2 cylinder_1_flow_head_I ≔ cylinder_1_input_diameter_min_val
   @cylinder_1_act0_3 cylinder_1_mode ≔ 0
   @cylinder_2_act0_0 cylinder_2_piston_position_O :∈ cylinder_2_cap_pos..cylinder_2_head_pos
   @cylinder_2_act0_1 cylinder_2_flow_cap_I ≔ cylinder_2_input_diameter_min_val
   @cylinder_2_act0_2 cylinder_2_flow_head_I ≔ cylinder_2_input_diameter_min_val
   @cylinder_2_act0_3 cylinder_2_mode ≔ 0
   @system_act_r5_0 system_connection_EVs_Doors_r6_cap_0 ≔ cylinder_0_input_diameter_min_val
   @system_act_r5_1 system_connection_EVs_Doors_r6_cap_1 ≔ cylinder_1_input_diameter_min_val
   @system_act_r5_2 system_connection_EVs_Doors_r6_cap_2 ≔ cylinder_2_input_diameter_min_val
   @system_act_r5_3 system_connection_EVs_Doors_r6_head_0 ≔ cylinder_0_input_diameter_min_val
   @system_act_r5_4 system_connection_EVs_Doors_r6_head_1 ≔ cylinder_1_input_diameter_min_val
   @system_act_r5_5 system_connection_EVs_Doors_r6_head_2 ≔ cylinder_2_input_diameter_min_val
   @system_act_r5_6 cylinders_read_r5 ≔ COMPONENTS_R5 × {0}
   @system_act_r5_7 system_control_r5_cap ≔ 0
   @system_act_r5_8 system_control_r5_head ≔ 0
 end
```

```
event GEV_0_environment extends GEV_0_environment
end

event system_connection_GEV_0_EVs extends system_connection_GEV_0_EVs
end

event valve_0_environment refines valve_0_environment
  where
    @grd0_0 valve_0_mode = 0
    @system_grd_r3_0 system_control_r2 = 2
    @system_valve_0_worked_grd valves_read_r3(ev0_r3) = 0
    @system_control_grd_r5_0 system_control_r5_cap = 0
  then
    @act0_0 valve_0_mode ≔ 1
    @act0_1 valve_0_control_I :∈ valve_0_CONTROL
    @act0_2 valve_0_flow_I ≔ system_GEV_0_EVs_connection_r3_0
    @system_valves_read_r4_0 valves_read_r3(ev0_r3) ≔ 1
    @system_act_r5_0 system_control_r5_cap ≔ 1
end

event valve_1_environment refines valve_1_environment
  where
    @grd0_1 valve_1_mode = 0
    @system_grd_r3_0 system_control_r2 = 2
    @system_valve_1_worked_grd valves_read_r3(ev1_r3) = 0
    @system_control_grd_r5_0 system_control_r5_head = 0
  then
    @act0_0 valve_1_mode ≔ 1
    @act0_1 valve_1_control_I :∈ valve_1_CONTROL
    @act0_2 valve_1_flow_I ≔ system_GEV_0_EVs_connection_r3_1
    @system_valves_read_r4_0 valves_read_r3(ev1_r3) ≔ 1
    @system_act_r5_0 system_control_r5_head ≔ 1
end

event valve_2_environment extends valve_2_environment
end

event valve_3_environment extends valve_3_environment
end

event GenericComponent_0_environement extends GenericComponent_0_environement
end

event system_connection_EVs_Doors_cap refines system_connection_EVs_Doors_cap
  where
    @system_grd_r4_0 valve_0_mode = 0
    @system_grd_r5_0 system_control_r5_cap = 1
  then
    @system_act_r5_0 system_control_r5_cap ≔ 2
    @system_act_r5_1 cylinders_read_r5 ≔ COMPONENTS_R5 × {0}
    @system_connection_act_r5_2 system_connection_EVs_Doors_r6_cap_0 ≔ valve_0_flow_O
    @system_connection_act_r5_3 system_connection_EVs_Doors_r6_cap_1 ≔ valve_0_flow_O
    @system_connection_act_r5_4 system_connection_EVs_Doors_r6_cap_2 ≔ valve_0_flow_O
end

event system_connection_EVs_Doors_head refines system_connection_EVs_Doors_head
  where
    @system_grd_r4_0 valve_1_mode = 0
```

```
  @system_grd_r5_0 system_control_r5_head = 1
 then
  @system_act_r5_0 system_control_r5_head ≔ 2
  @system_act_r5_1 cylinders_read_r5 ≔ COMPONENTS_R5 × {0}
  @system_connection_act_r5_2 system_connection_EVs_Doors_r6_head_0 ≔ valve_1_flow_O
  @system_connection_act_r5_3 system_connection_EVs_Doors_r6_head_1 ≔ valve_1_flow_O
  @system_connection_act_r5_4 system_connection_EVs_Doors_r6_head_2 ≔ valve_1_flow_O
end

convergent event cylinder_0_environment
 any system_control_cap_new_r5 system_control_head_new_r5
 where
  @grd0_0 cylinder_0_mode = 0
  @system_control_grd_r5_0 system_control_r5_cap = 2
  @system_control_grd_r5_1 system_control_r5_head = 2
  @system_grd_r5_2 cylinders_read_r5[COMPONENTS_R5∖{cylinder_0_r5}] = {1} ⇒
                                              system_control_cap_new_r5 = 0 ∧ system_control_head_new_r5 = 0
  @system_grd_r5_3 ¬cylinders_read_r5[COMPONENTS_R5∖{cylinder_0_r5}] = {1} ⇒
                                              system_control_cap_new_r5 = 2 ∧ system_control_head_new_r5 = 2
  @system_grd_r5_4 cylinders_read_r5(cylinder_0_r5) = 0
 then
  @act0_0 cylinder_0_mode ≔ 1
  @act0_1 cylinder_0_flow_cap_I ≔ system_connection_EVs_Doors_r6_cap_0
  @act0_2 cylinder_0_flow_head_I ≔ system_connection_EVs_Doors_r6_head_0
  @system_act_r5_0 system_control_r5_cap ≔ system_control_cap_new_r5
  @system_act_r5_1 system_control_r5_head ≔ system_control_head_new_r5
  @system_act_r5_2 cylinders_read_r5(cylinder_0_r5) ≔ 1
end

convergent event cylinder_1_environment
 any system_control_cap_new_r5 system_control_head_new_r5
 where
  @grd0_0 cylinder_1_mode = 0
  @system_control_grd_r5_0 system_control_r5_cap = 2
  @system_control_grd_r5_1 system_control_r5_head = 2
  @system_grd_r5_2 cylinders_read_r5[COMPONENTS_R5∖{cylinder_1_r5}] = {1} ⇒
                                              system_control_cap_new_r5 = 0 ∧ system_control_head_new_r5 = 0
  @system_grd_r5_3 ¬cylinders_read_r5[COMPONENTS_R5∖{cylinder_1_r5}] = {1} ⇒
                                              system_control_cap_new_r5 = 2 ∧ system_control_head_new_r5 = 2
  @system_grd_r5_4 cylinders_read_r5(cylinder_1_r5) = 0
 then
  @act0_0 cylinder_1_mode ≔ 1
  @act0_1 cylinder_1_flow_cap_I ≔ system_connection_EVs_Doors_r6_cap_1
  @act0_2 cylinder_1_flow_head_I ≔ system_connection_EVs_Doors_r6_head_1
  @system_act_r5_0 system_control_r5_cap ≔ system_control_cap_new_r5
  @system_act_r5_1 system_control_r5_head ≔ system_control_head_new_r5
  @system_act_r5_2 cylinders_read_r5(cylinder_1_r5) ≔ 1
end

convergent event cylinder_2_environment
 any system_control_cap_new_r5 system_control_head_new_r5
 where
  @grd0_0 cylinder_2_mode = 0
  @system_control_grd_r5_0 system_control_r5_cap = 2
  @system_control_grd_r5_1 system_control_r5_head = 2
  @system_grd_r5_2 cylinders_read_r5[COMPONENTS_R5∖{cylinder_2_r5}] = {1} ⇒
                                              system_control_cap_new_r5 = 0 ∧ system_control_head_new_r5 = 0
  @system_grd_r5_3 ¬cylinders_read_r5[COMPONENTS_R5∖{cylinder_2_r5}] = {1} ⇒
```

```
                                    system_control_cap_new_r5 = 2 ∧ system_control_head_new_r5 = 2
    @system_grd_r5_4 cylinders_read_r5(cylinder_2_r5) = 0
   then
    @act0_0 cylinder_2_mode ≔ 1
    @act0_1 cylinder_2_flow_cap_I ≔ system_connection_EVs_Doors_r6_cap_2
    @act0_2 cylinder_2_flow_head_I ≔ system_connection_EVs_Doors_r6_head_2
    @system_act_r5_0 system_control_r5_cap ≔ system_control_cap_new_r5
    @system_act_r5_1 system_control_r5_head ≔ system_control_head_new_r5
    @system_act_r5_2 cylinders_read_r5(cylinder_2_r5) ≔ 1
  end
…
end
```

TURKU
CENTRE *for*
COMPUTER
SCIENCE

Joukahaisenkatu 3-5 A, 20520, Turku, Finalnd | www.tucs.fi

University of Turku
Faculty of Mathematics and Natural Sciences
- Department of Information Technology
- Department of Mathematics

Turku School of Economics
- Institute of Information Systems Sciences

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

ISBN 978-952-12-3311-1
ISSN 1239-1891