Runtime Contracts Checker Specification

Miren Illarramendi, Leire Etxeberria, Xabier Elkorobarrutia, and Goiuria Sagardui

Embedded Systems Research Group,
Mondragon Goi Eskola Politeknikoa (MGEP) Arrasate-Mondragon, Spain
{millarramendi,letxeberria,xelkorobarrutia,gsagardui}@mondragon.edu

1 Runtime Contract Checker Specification

1.1 Specification using Safety Contracts

We need to prove that the composite implementation of the system guarantees system level contracts at runtime.

In our case, a safety contract specifies properties related to the internal behavior of the SW components that are part of the system in terms of their UML-SM model (active states). That is what we call a state-based safety contract.

In our approach a system (Sys) may be composed of subsystems (that could be further decomposed) and primitive components (C) that can not be further decomposed. Furthermore, the primitive components have a behavior specified using a state machine. A system (Sys) is composed of at least one primitive Component (C), i.e., $Sys = \{C_1, C_2, ...C_{nc}\}$ where nc is the total number of primitive components (Cs) in the system (Sys). Accordingly, a C in our context is state-based. Each of these Components has a set of states (S), i.e., $C_i = \{S_1, S_2, ...Sns_{C_i}\}$ where ns_{C_i} is the number of states that comprise the i-th C.

Let us denote the active state of a component (C) at a discrete time point as $C_i.S_j$, the state S_j being any of the states the C component ($S_j \in C_i$) may have. Safety requirement will be allocated to the system. And a safety requirement may be satisfied by a safety contract or a set of safety contracts. A safety contract will be related to the states of the components involved in the contract. A grammar with regular expressions is used to specify what to check. The relevant syntax of this grammar has been summarized:

$$\begin{aligned}
\textit{contract} &:= \textit{constraint} \mid \textit{timedConstraint};\\
\textit{constraint} &:= \textit{condition} \ \ \textbf{implies} \ \ \textit{condition};\\
\textit{condition} &:= \textit{activeState} \mid \textbf{not} \ \ \textit{condition} \mid (\textit{condition}) \mid\\
&\quad\ \textit{condition} \ \ \textbf{or} \ \ \textit{condition} \mid\\
&\quad\ \textit{condition} \ \ \textbf{and} \ \ \textit{condition};\\
\textit{timedConstraint} &:= \textit{constraint} \ \ \textbf{in} \ \ \textit{timeUnits};\\
\textit{activeState} &:= ComponentName.StateName;
\end{aligned}$$

With this grammar, it is possible to specify constraints regarding the active states of components of the system. For instance, we can specify that the active state of the C_{Door} must be S_{Closed} when the active state of the $C_{Traction}$ is S_{On}.

$$C_{Traction}.S_{On} \quad \textbf{implies} \quad C_{Door}.S_{Closed}$$

We can also use logical operators as well as parentheses to build more complex conditions. For example, when C_{Door} is in the S_{Closed} and $C_{Traction}$ is in the S_{On}, then the $C_{Obstacle}$ must be in S_{NoObst}.

$$C_{Door}.S_{Closed} \quad \textbf{and} \quad C_{Traction}.S_{On} \quad \textbf{implies} \quad C_{Obstacle}.S_{NoObst}$$

In the previous examples, we referred to the active states of the components in the current time point. We can also specify constraints regarding future states of components using timedConstraints. A timedConstraint specifies a constraint that must become true in the interval between the current time and the current time plus timeUnits. For example, when the active state of the $C_{Obstacle}$ is S_{Obst} then the active state of the C_{Door} must be $S_{Opening}$ and not any other state of the C_{Door} component in 2000 milliseconds time.

$$C_{Obstacle}.S_{Obst} \quad \textbf{implies} \quad C_{Door}.S_{Opening} \quad \textbf{in} \quad 2000$$

2 Case Study: A Train Door Controller.

In this case study, a door control management performed by a train control and monitoring system (TCMS) was considered. The TCMS is a complex distributed system that controls many subsystems such as the door control, traction system, air conditioning, etc. The system level requirements concerning the operation of opening and closing of doors are satisfied by the following components:

- `TCMS` component decides whether to enable or disable the doors considering the driver's requests and the train movement. Thus, doors must be enabled before they can be opened;
- `Door` component controls and commands the opening and closing of a door;
- `Traction` component controls and commands the train movement;
- `Obstacle Detection` component manages the obstacle detection in the door.

The case study concerns a real industrial system where some simplifications were made. Specifically, the interaction with other components of the TCMS, the dependencies with other subcomponents and their communication were omitted.

Fig. 1 shows the UML-SM of the `DoorController`, `ObstacleDetector` and `Traction`.

The following safety requirements are selected in the context of this case study:

1. **SR1.** When door is open or opening or closing, the traction is off.
2. **SR2.** When door is closed, there must not be obstacles detected.
3. **SR3.** When traction is on, and a passenger presses the opening button, the door must remain closed.
4. **SR4.** When traction is on, there must not be obstacles detected.

The next step is to express these safety requirements as state-based safety contracts, and the result is shown in the listing 1.1. Each system level rule in this listing corresponds to each one of the aforementioned requirements.

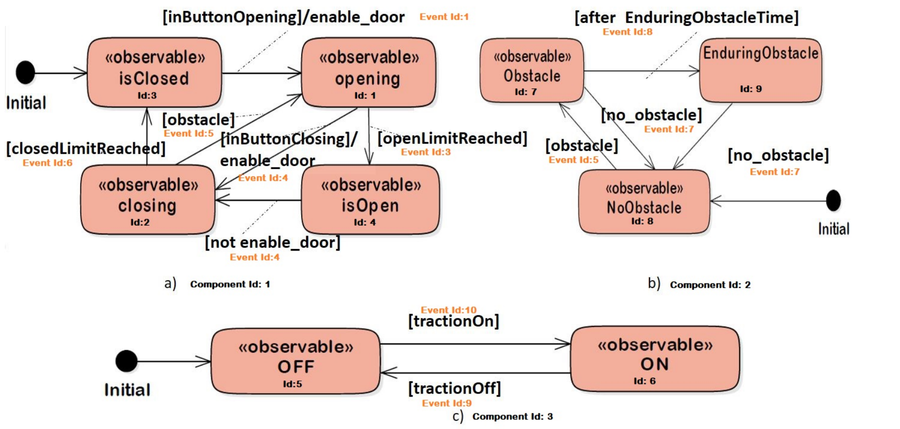

Fig. 1. UML-SM Diagrams of a)`DoorController` b)`ObstacleDetector` and c)`Traction`

Listing 1.1. Safety Requirements expresed as Safety Contracts

```
SC1:CDoor.Sopen or CDoor.Sopening or CDoor.Sclosing implies CTraction.
    Soff
SC2:CDoor.Sclosed implies Obst.Sno_obst
SC3:CTraction.Son implies CDoor.Sclosed
SC4:CTraction.Son implies CObst.Sno_obst
```

At this point, $safeStop$ processes are defined for each of the contracts. In this use case, one simple $safetStop$ process was implemented for all of the contracts: when violating any of the contracts the traction is sent to the OFF state to stop the train, the doors are closed (isClosed state) and the obstacle detector is sent to the NoObstacle state.