Monitoring and Enactment with Reactive Event Calculus

In Chap. 13, we have shown how sciff can be used to perform run-time verification of a running execution with respect to some ConDec model. When providing execution support, it is not only important to offer compliance verification facilities, but also to give a constant feedback to the interacting entities, keeping them informed about the evolving state of affairs and reporting undesired situations. This task is called monitoring and is illustrated in Fig. 14.1. Monitoring aims at dynamically observing the behavior of interacting entities, tracking its impact on the monitored specification and capturing violations without terminating the computation; the detection of a violation could lead to generate a corresponding alarm, to warn the system administrator or even to start a special course of interaction, aimed at fixing the violation.

In this chapter, we show how a reactive form of the Event Calculus (EC) [146] can be encoded as a SCIFF-lite program, enabling the possibility of

- monitoring ConDec optional constraints;
- introducing compensation constructs in ConDec, modeling business constraints that express which countermeasures should be taken when an optional constraint is violated, and that are enforced only in such an exceptional situation;
- tracking the evolution of constraints’ states as events occur.

The latter topic provides the basis for supporting the enactment of ConDec models, which is discussed in the last part of the chapter.

14.1 Monitoring Issues and SCIFF

When it comes to the monitoring problem, the sciff proof procedure has two limits. First of all, one could think that pending expectations are a suitable candidate for representing the current state of affairs, provided that suitable
assumptions, such as ascending order of event occurrences, hold\(^1\). However, there are situations in which the generated expectations are not significant. A typical case is when the model contains branching constraints, i.e., it leaves open alternatives to the interacting entities. The depth-first nature of the current sciff implementation drives the proof procedure to investigate one alternative at a time, and there is no guarantee that the selected alternative is the same as the one chosen by the interacting entities. When such a mismatch exists, sciff will switch to the right alternative as soon as a violation related to the other one is detected. Thus, there is a lapse of time during which the pending expectations shown by sciff do not fit with the actual behavior, and therefore they should not be exposed to the interacting entities.

**Example 14.1.1 (Run-time verification of branching constraints).** A care-flow protocol involves the following business constraint: when a sample is sent to the laboratory, then either an error concerning the sample must be communicated by the laboratory within 5 time units, or a the analysis result must be produced within 15 time units. It can be modeled in ConDec by means of a metric branching response constraint, mapped onto CLIMB as follows:

\[
\begin{align*}
 t_{IC} = \begin{cases} 
 \text{send sample} & (0,5) \\
 \text{sample error} & (0,15) \\
 \text{produce result} 
\end{cases} \\
 H(\text{exec} (\text{send sample}), T_s) \rightarrow E(\text{exec} (\text{sample error}), T_e) \\
 & \land T_e > T_s \land T_e < T_s + 5 \\
 \lor E(\text{exec} (\text{produce result}), T_r) \\
 & \land T_r > T_s \land T_r < T_s + 15
\end{align*}
\]

We now consider a specific execution of the care-flow protocol. At time 9, a sample is sent to the laboratory. The integrity constraint is triggered, and one

\(^1\) If events do not occur in ascending order, then it is inevitable that sciff revises the computed results, because it becomes aware of new information about the past after having already processed it.