The PEPL Language

The PEPL language builds on top of DCR Graph semantics, with additional extensions. As such, its primary elements are events and processes.

A PEPL module is a collection of processes, and a process is a collection of events and processes.

Processes and events can be parametrized with data - in the current implementation the allowed types are strings, numbers, and process references.

Following DCR Graphs semantics, an event is enabled when its conditions are satisfied and milestones are met. The former are events that must be either excluded from execution or have been already executed. The latter are events that must be either excluded or currently not required as response.

The execution of an event has the effect of performing its actions: these can be requiring an action, and including and excluding events. Processes are spawned when one of their initial events is executed.

Requiring an event as action put it into the process to-do list. As in DCR Graph semantics, a process will not be accepting when there exists at least one included response.

An event required as auto response will be executed automatically as soon as it is enabled.

Dynamic event exclusion and inclusion effectively removes – respectively adds – the event from the execution: an excluded event cannot be executed and, e.g., it is not considered as a condition.

Theory - Dynamic Condition Response Graphs (DCR Graphs)

The foundation of PEPL is the recently developed Dynamic Condition Response (DCR) Graphs model aimed at specification, execution and verification of pervasive, event-based workflow and business processes. A key feature of DCR Graphs is that the state of a process is entirely driven by which events have happened in the past, which events (responses) are required to happen for the process to make progress, and which events are currently included in the process. This means in particular that there is no synchronization via shared variables nor program pointer.

To serve as a basis for a general event-based programming language, we extend the DCR Graphs semantics to allow 1) events and relations to be parameterized with data values, 2) sub processes to be dynamically created, and 3) specification of events to be executed by a processor. Such extensions were needed in order to overcome the limitations imposed by a high-level modeling language such as DCR Graphs.

The main benefit of having a formal foundation is that we are able to use the results of the theoretical model in the language implementation. For example, in [1] a general distribution technique for DCR Graphs is presented. Furthermore, in [2], it is shown that the execution semantics of a finite DCR Graph can be formally represented as finite Büchi-automaton and verified using the SPIN model-checker.