Towards a Programming Language for Declarative Event-based Context-sensitive Reactive Services

Søren Debois Thomas Hildebrandt Raghava Rao Mukkamala Francesco Zanitti
*{debois, hilde, rao, frza}@itu.dk
IT University of Copenhagen
Programming, Logic and Semantics Group
Rued Langgaards Vej 7, DK-2300 Copenhagen S, Denmark

Abstract

We present ongoing work on a new declarative and purely event-based programming language, tentatively named *DECoReS*, for *Declarative Event-based Context-sensitive Reactive Services*. The language is based on an extension of the recently developed declarative Dynamic Condition Response (DCR) Graphs model for concurrent processes, which generalizes the classical model of prime event structures to a systems model in which infinite behavior and progress constraints can be represented by finite structures. To give semantics for the DECoReS programming language the DCR Graph model is extended with parametrized events, auto events, sub processes, time and exception handling.

1 Introduction

The Dynamic Condition Response (DCR) Graph model has been developed as part of the CosmoBiz [1] and TrustCare [2] research projects with the goal to provide a formal foundation for adaptable and flexible pervasive workflow processes and services as found e.g. within the healthcare domain.

The development of the DCR Graph model takes its outset in the declarative Process Matrix workflow model [7, 8] implemented by our industrial partner Resultmaker and generalizes the classical event structure model for concurrency [10, 11] to allow for *finite* descriptions of infinite behavior (also referred to as a systems model [9]) and specification of progress constraints.

The key elements of a DCR Graph is a set of (labelled) events (e.g. representing executions of human activities in a workflow or service requests and responses), two dual relations between events referred to as the *condition* and *response* relations respectively, and two relations for *dynamically* including and excluding events from the process.

An unconstrained event can happen at any time and any number of times as long as it is included in the process. The included conditions of an event $\in$ are the events that must happen at some point before event $\in$ happens. Dually, the responses of an event $\in$ are the events that must happen at some point after event $\in$ happens, or infinitely often become excluded. Despite its simplicity, the DCR Graph model can express all ω-regular languages and thus in particular all processes that can be specified in Linear-time Temporal Logic (LTL).

Moreover, the model allows for an intuitive operational semantics and effective execution expressed by a notion of markings of the graphs. A marking is given by three sets of events $(\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$, where Ex is the set of previously executed events, Re the activities required to be executed in the future (i.e. pending responses), and In is the currently included activities.

*Authors listed alphabetically. This research is supported by the Danish Research Agency through a Knowledge Voucher granted to Exformatics (grant #10-087067, www.exformatics.com), the Trustworthy Pervasive Healthcare Services project (grant #2106-07-0019, www.trustcare.eu) and the Computer Supported Mobile Adaptive Business Processes project (grant #274-06-0415, www.cosmobiz.dk). This work funded in part by the Danish Research Agency (grant no.: 2106-080046) and the IT University of Copenhagen (the Jingling Genies projects).

The DCR Graph model has been applied to healthcare and cross-organizational case management processes identified in case studies carried out jointly with industrial partners [3,4]. The case studies led to extensions of the core model to allow nested events and a new relation between events describing a *milestone* relation [5]. An event `e` is not enabled for execution if any of its included milestones is in the set Re of pending responses. The case studies also revealed the need for distributed implementations, which led to the development of a general technique for distributing a DCR Graph based on a notion of projections [6].

Below we exemplify current work on defining and implementing a new declarative and purely event-based language based on DCR Graphs, tentatively named *DECoReS*, for Event-based Context-sensitive Reactive Services. To support the formal semantics of DECoReS, we propose extending the model of DCR Graphs with *parametrized* events, sub processes, *automatic* events, time and exception handling.

The languages and extensions are illustrated by the following example workflow process adapted from [7, 3]. The process consists of five events: The prescription of medicine and signing of the prescription by a doctor (represented by the events `prescribe` and `sign` respectively), a nurse giving the medicine to the patient (represented by the event `give`, and the nurse indicating that he does not trust the prescription (represented by the event `distrust`) and the doctor removing the prescription, represented by the event `remove`.

```
treatment process{
  doctor may prescribe<$id, $med, $qty> {
    response: administer<$id,$med,$qty>
  }
  administer<$id, $med, $qty> process
  {
    doctor must sign { exclude: remove }
    nurse must give {
      condition: Executed(sign) &
                 not Executed(remove) &
                 not Response(sign)
      exclude: sign, give, distrust, remove
    }
    nurse may distrust {
      response: sign
      include: remove
      exclude: give
    }
    doctor may remove {
      exclude: sign,give,distrust,remove
    }
  }
}
```

To capture that every prescription event `prescribe` leads to the possible execution of a "fresh" set of events `sign`, `give`, `distrust` and `remove`, we propose as the first extension of DCR Graphs to allow *sub process events* that instantiate a DCR Graph as a sub process when they are required as a response. This allows us to group the four events inside the sub process event `administer`.

The result is a process model which is more expressive than ω regular languages but still have an intuitive operational semantics as dynamically unfolding graphs.

A third proposed extension is to allow events to be called automatically. Such events are called "auto-events" and they can be emitted in order to communicate to the external world some important state of the process.

Finally, we work on adding time deadlines to the constraints, i.e. making it possible to specify that a given response must happen within a given time interval. This then again leads to the need for handling violations of such constraints. The modified process below illustrates how time constraints and exception handling can be added to the language, while also giving a small example of how the language can be used for implementing context-sensitive services.

```
doorsensor process {
  door may open {
    response: close within 15s throw door_left_open
  }
  door may close
  door_left_open process {auto leftopen}
}
```

In the above sample process, we model a process that senses when a door is being opened and closed. This is conveniently written using the role `door` for sensor at the door. The timing constraint `within 15s` specifies that in response to an `open` event, a `close` one must follow within 15 seconds, otherwise an instance of the sub process `door_left_open` will be created, which executes an autoevent `leftopen`.

Informally, an event that is required as a `response` can be annotated with a timing constraint interval `in Start within End` must happen after `Start` time units but before `End` time units; omitting the `Start` constraint is a shortcut for `in 0 within End`. Dually, a timed `condition` event, annotated with a time constraint `since Start within End`, must have happened at least `Start` time units before and at most `End` time units before. Omitting the `Start` constraint is a shortcut for `since 0 within End`.

Lastly, timed responses can be annotated with an `throw Event` construct, which requires `Event` to happen as a response, if the response do not happen within the required interval. Similarly, conditions may throw such exception events if an event happen and the condition is not satisfied.

References

[1] Thomas Hildebrandt. Computer supported mobile adaptive business processes (CosmoBiz) research project. Webpage, 2007. `http://www.cosmobiz.org/`.

[2] Thomas Hildebrandt. Trustworthy pervasive healthcare processes (TrustCare) research project. Webpage, 2008. `http://www.trustcare.dk/`.

[3] Thomas Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Declarative modelling and safe distribution of healthcare workflows. In *International Symposium on Foundations of Health Information Engineering and Systems*, Johannesburg, South Africa, August 2011.

[4] Thomas Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Designing a cross-organizational case management system using dynamic condition response graphs. In *Proceedings of IEEE International EDOC Conference*, 2011. to appear.

[5] Thomas Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Nested dynamic condition response graphs. In *Proceedings of Fundamentals of Software Engineering (FSEN)*, April 2011. to appear.

[6] Thomas Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Safe distribution of declarative processes. In *9th International Conference on Software Engineering and Formal Methods (SEFM) 2011*, 2011. to appear.

[7] Karen Marie Lyng, Thomas Hildebrandt, and Raghava Rao Mukkamala. From paper based clinical practice guidelines to declarative workflow management. In *Proceedings of 2nd International Workshop on Process-oriented information systems in healthcare (ProHealth 08)*, pages 36–43, Milan, Italy, 2008. BPM 2008 Workshops.

[8] Raghava Rao Mukkamala, Thomas Hildebrandt, and Janus Boris Tøth. The resultmaker online consultant: From declarative workflow management in practice to LTL. In *Proceeding of DDBP*, 2008.

[9] Vladimiro Sassone, Mogens Nielsen, and Glynn Winskel. A classification of models for concurrency. In *Proceedings of CONCUR'93*, volume 715 of *LNCS*, 1993.

[10] Glynn Winskel. Event structures. In Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Advances in Petri Nets*, volume 255 of *Lecture Notes in Computer Science*, pages 325–392. Springer, 1986.

[11] Glynn Winskel and Mogens Nielsen. Models for concurrency. pages 1–148, 1995.