Distributed Systems Specification and Adaptation : A Petri net-based Proposal*

Nasreddine Aoumeur
Institut für Technische Informationssysteme
Otto-von-Guericke-Universität Magdeburg
Postfach 4120, D–39016 Magdeburg
E-mail: aoumeur@iti.cs.uni-magdeburg.de
Fax: ++49-391-67-12020

Zusammenfassung

Most of present-day systems are data-intensive and fully distributed. Moreover, such systems are often layered into several towers depending on the services they are offering. This latter characterization induce challenging problems to designers of such systems, including how should new requirements be dynamically introduced, modified, interacted with existing ones? and so on. In this paper we propose a true-concurrent object Petri-net-based framework for addressing such challenges. The model, referred to as CO-NETS, integrates object-orientation concepts with modularity aspects into an appropriate algebraic Petri nets variant, and it is semantically interpreted in rewrite logic. This integration suffices for adequately addressing the data-intensive and full distribution. For dynamically introducing and modifying any requirement, we soundly extend this integration with some reflection capabilities. All the proposed ideas are illustrated using a system with several lifts.

1 Introduction

Most of present-day software systems are characterized as reactive systems with large amount of data and complex and fully distributed behaviour. Moreover, besides these key aspects, system designers are also facing another crucial challengings, namely the dynamic evolution of such systems in terms of different services they may offer to their users. In fact in most cases, real-world complex systems, such as telecommunication applications, are structured into a hierarchy of layers, each one is mainly distinguished by the features that may offer. Among the conceptual challenges involving features are particularly: how to dynamically express them, to introduce them, to modify them and how to coherently integrating them with each others ?.

Although the discipline for coping with dynamic evolution is quite new—often referred to as feature interaction— there have been severals interesting proposals, in terms of appropriate formal languages [5] [3]. However, due to the complexity of the problem we are far from a widely accepted framework. Moreover, most of existing proposals have concentrated more on the features themselves, while ignoring or disadvantaging the structural and behavioural aspects of the system. With the aim to overcome such limitations, we propose in this paper an appropriate framework that aims to cover most of the mentioned aspects. The model, referred to as CO-NETS [2], consists in a sound and complete integration of object oriented constructions with some modularity concepts into a well-suited variety of algebraic Petri nets. The model is semantically

*The complete version of the paper is to appear as a print.

interpreted in rewriting logic. Moreover, recently we have enhanced this proposal with some reflection capabilities allowing runtime modification of existing behaviour [1].

The rest of this paper is organized as follows. The next section introduces and specify a simplified version of the lift system using the Co-nets framework, that due to space limitation we assume the reader is more or less familiar with. In the third section, we addresses the problem of how features may be dynamically manipulated. We conclude this paper with some remarks and outlying our future work.

2 Specification of the lift system using the Co-nets approach

In the following, first, we informally present the running case study, afterwards we present its data level and template signature.

2.1 Informal description of the case study.

As we pointed out the case study we use throughout this paper for illustrating the developed ideas consists in a distributed system with several lifts. Thinking in an object oriented way [6] [4], the simplified (and initial) version we treat in this paper may be informally described as follows:

- Each lift is structurally characterized by its identity (name or number), by the current floor it is on (shortly `Cur_F`), by its state expressing if it is idle, going up or going down (shortly `St`), by the current state of its door (`St`), and by its current weight (`Wg`).
- The operations or methods that may offer each lift are the following. The lift can open or close its door, go to any called floor with respect to a given direction (up or down) and stop at intermediate floors if there have been a corresponding call meanwhile. Moreover, each lift is able to cancel calls from the same floor. Also, to make a distinction between a request inside the lift (i.e. a goto) or a call from the outside, we use for both a method we denote by `Goto_F(Id-lift, dest-floor, call-or-goto)`; where the value of the parameter `call-or-goto` allows to distinguish between the two kinds of request. Finally, for technically keeping trace of a given call while serving its intermediate floors, we introduce another variant of goto, that is `Goto_F(Id-lift, dest-floor, direction)`, where `direction` stands for the direction (i.e. Up or Dw) and `dest-floor` is the selected floor (which should be the max (resp. min) when the direction is Up (resp. Dw) from current calls).

Data specification for the lift signature. The different data sorts we need for the template signature are the following. For describing different floor levels, we use the sort `Floors` whose elements are natural constants (0,1, 2, .., k). For capturing different states a given lift may be in, we use the `StateF` whose elements are also constants, namely (the self-explained ones) `idle` or `Up` or `Dw`. The doors state is captured by a data sort `Door`, with two values: `op` for open and `cl` for closed. Also, we denote by `Wmx`, the maximal weight tolerated by each lift, which has to be a positive real constant. For distinguishing between a call from outside the lift and goto from the inside, we use a data sort denoted `Call-or-Goto` with two values: `out` for a call from the outside and `in` for a goto from the inside of the lift. Besides these data sorts, for capturing correctly the behaviour of a given lift we present in the next subsection, we need a function we called `less(K1, K2)`, where `K1` denote final destination of the lift and `K2` the current floor of the lift. This function has to return the first minimal next floor number between $K1$ and $K2$ which is currently requested. The two equations corresponds to the two cases : the lift is going up (i.e. $K1 > K2$) or the lift is going down (i.e. $K1 < K2$). All in all this level is:

```
obj Lift-Data is
  protecting Real+ nat .
  sort Door StateF Call-or-Goto.
  op 0, 1, 2, 3, 4, 5, ..  , k :  → Floors .
  op idle, Up, Dw :  → StateF .
  op op, cl :  → Door .
  op in, out :  → Call-or-Goto .
  opoplessidle, Up, Dw :  → StateF .
  op Wmx :  → Real+ .
  vars O : Call-or-Goto .
  vars K1, K2 :  Floors .
  eq less(K1, K2) == if K1 > K2 then K2 + 1 or
      K2+2 or ...  or K1 - 1 endif .
  eq less(K1, K2) == if K1 < K2 then K2 - 1 or
      K2 - 2 or ...  or K1 - 1 endif .
endo.
```

The lift template signature. Using this data level and the informal OO description, the template signature corresponding to the lift can be given as follows.

```
obj Lift is
  extending Object-State .
  protecting Lift-Data .
  sort Id.Lift < OId .
  sort GOTO TEST LIFT .
  (* the Lift object state declaration *)
   op ⟨_|Cur_F : _, St : _, Dr : _, Wg : _⟩:  Id.Lift
        Floors  StateF  DoorSt  Real+ → Lift .
(* Messages declaration *)
   op Goto_F : Id.Lift  Floors  Call-or-Goto → GOTO .
   op Goto_F : Id.Lift  Floors  StateF  → GOTO .
   op TestF : Id.Lift  Floors  Floors  nat → TEST .
  vars L : Id.Lift .
  vars S, D : StateF .
  vars W, W' : Real+ .
  vars K, K1, K2, K' : Floors .
endo.
```

We note that all the specified variables will be used in the corresponding nets later. ■

2.2 Template specification

Given a template signature denoted by TS that captures the structural aspects of a distributed system, its behaviour is constructed by associating a Co-NET with this signature—leading to the notion of template specification that we denote by $SP = \prec TS, Net \succ$. Informally speaking, the net associated with a given template signature is constructed as follows.

- The places of the net are precisely defined by associating with each message generator one ‘message’ place. Also, with each object state sort an ‘object’ place is associated. We denote the set of all places by P.

- Transitions, which may include conditions, reflect the effect of messages on object states (i.e. method body) to which they are addressed. Note that, for distinguishing between local messages and external ones we draw the later with bold lines.

2.2.1 The object net of the lift system

Following these very simplified rules for deducing an object net from a given template, in Figure 1 we depict the corresponding net modeling the dynamic of a system with several lifts. First with the object sort `Lift` we have associated a corresponding place which contains the current component-state of each lift. Also, with the message sort `GOTO` we have associated a corresponding place.

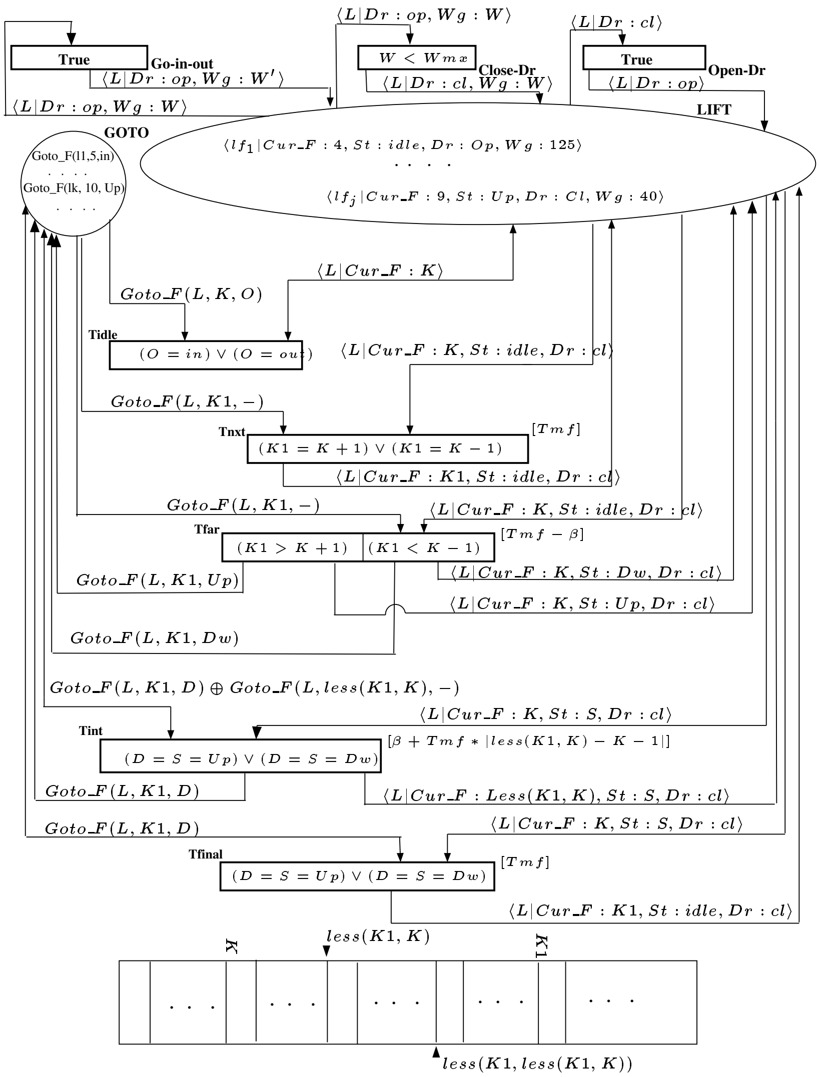

Abbildung 1: The Lift Component as a CO-NET.

3 Conclusions

We presented in this paper a general-purpose framework, referred to as CO-NETS, particularly suited for specifying complex distributed systems, and for dynamically manipulating their features. Methodological this framework may be regarded as three layer-based one. The first layer is a sound integration object oriented abstractions mechanism with modularity constructs into an appropriate variety of algebraic Petri nets. The second layer is meta-level one, and it allows for dynamically creating, modifying and/or deleting features while the system is still running. The third layer takes profit of the reflection capabilities of rewrite logic—as semantics for CO-NETS behaviour—for dynamically composing and interacting strategies, as appropriate and complex features.

We have illustrated this framework using a non trivial specification of a variant of a system with several lifts. This case study shown us in particular the suitability of this framework for dealing complex real-world distributed systems and their features. However, after achieving this first step we are conscious that much more work remains ahead. Particularly, we are planning to extend this case study to more complex one. Also, we are focusing on the integration of temporal aspects for verifying, and not only validating, system properties.

Literatur

[1] N. Aoumeur. Specifying Distributed and Dynamically Evolving Information Systems Using an Extended CO-NETS Approach. In G. Saake, K. Schwarz, and C Türker, editors, *Transactions and Database Dynamics*, volume 1773 of *Lecture Notes in Computer Science, Berlin*, pages 91–111. Springer-Verlag, 2000. *Selected papers from the 8th International Workshop on Foundations of Models and Languages for Data and Objects, Sep. 1999, Germany.*

[2] N. Aoumeur and G. Saake. Towards an Object Petri Nets Model for Specifying and Validating Distributed Information Systems. In M. Jarke and A. Oberweis, editors, *Proc. of the 11th Int. Conf. on Advanced Information Systems Engineering, CAiSE'99*, volume 1626 of *Lecture Notes in Computer Science*, pages 381–395. Springer-Verlag, 1999.

[3] M. Heissel. Detecting Features Interactions—A Heuristic. In *Proc. of the 1st FIREworks Worshop*, pages 30–48, Magdeburg, Germany, 1998. Technical Report, Preprint Nr. 10, Computer Science Department, University of Magdeburg.

[4] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object Oriented Modeling and Design.* Prentice Hall, Englewood Cliffs, 1991.

[5] M. Ryan. Features-oriented programming : A case study using the SMV language. Technical report, School of Computer Science, University of Birmingham, 1997.

[6] P. Wegner. Concepts and paradigms of Object-Oriented Programming. *OOPS Messenger*, 1:7–87, 1990.