UL^{++} : an Object Oriented Language for Hiearchical Modelling and Analysis

Alexandre David — adavid@student.docs.uu.se
Justin Pearson — justin@docs.uu.se
Wang Yi — yi@docs.uu.se

Uppsala University
Department of Computer Systems

April 3, 1999

Abstract

It is our experience with model-checkers like Uppaal that a large part of the total time in most of the case studies is spent on constructing the system model, i.e., modeling. This has lead to a strong demand for a modeling language with features supporting abstractions such as hierarchical descriptions of models. There have been several modelling languages e.g. State-Chart for Statemate, developed particularly for modelling and simulation. However, rather little work has been done on static analysis for such languages using information such as types, hierarchical structures etc, to guide verification.

In this talk, we present an ongoing work to develop an object–oriented modelling language UL^{++} as well as a graphical user interface for the tool Uppaal. The language is designed to support two types of hiearchical structures as illustrated in Figure 1 :

- the system hierarchy where a system consists of (nested) concurrent processes (networks of automata).
- the state hierarchy where a process consists of (nested) super-states (modes and states).

We present an object model, which allows systems to be described in a uniform manner. There are two main sorts of objects: control objects to represent control stuctures for automata and data objects for represent data structures. Data objects are divided into static (integers, booleans and arrays) and dynamic (clocks and channels) data objects. The complete object hierarchy is depicted in fig. 2. A very important feature of UL^{++} is the module concept: objects are closed and ther content is not accessible except by specified connections. Connections between objects are of two forms :

- **control interfaces** for superstates, which provide entry and exit points, visible from outside;

- **data interfaces** to instanciate objects with the data of a container object to work on.

UL^{++} defines an intuitive graphical syntax, close to statecharts, but with the control interface concept. Figure 1 gives an example of the graphical syntax. The textual syntax offers object-oriented capabilities such as extensions.

The semantics is given by four inductive rules which give the operational semantics in terms of transitions. The rules give the unfolding of abstract transitions, between superstates, in terms of concrete state transitions.

As future work, we plan to utilize the hierachical information in order to guide the state–space exploration algorithm that the UPPAAL engine is based on and to introduce *abstraction* in the UPPAAL simulator and diagnostic trace–generator.

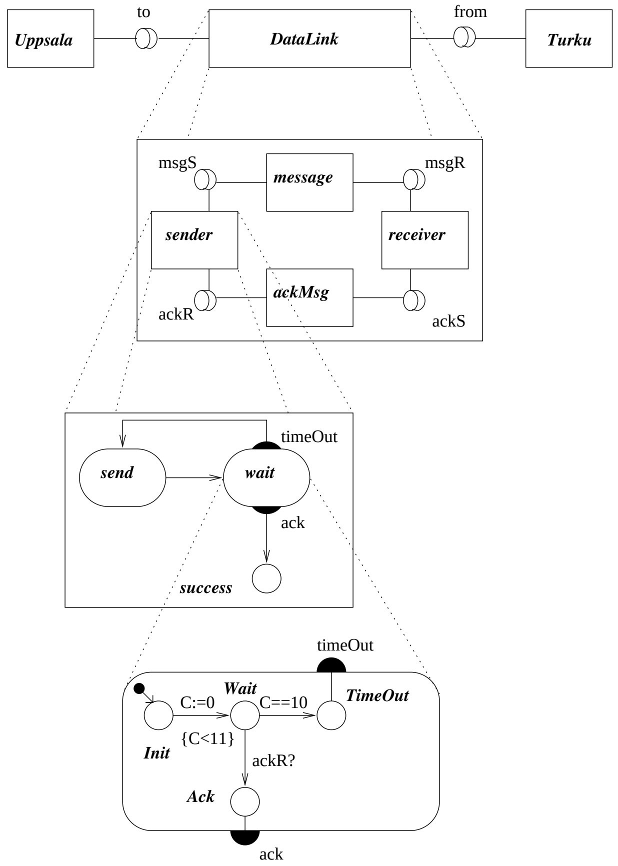

Figure 1: An example of UL^{++} hierarchial structures.

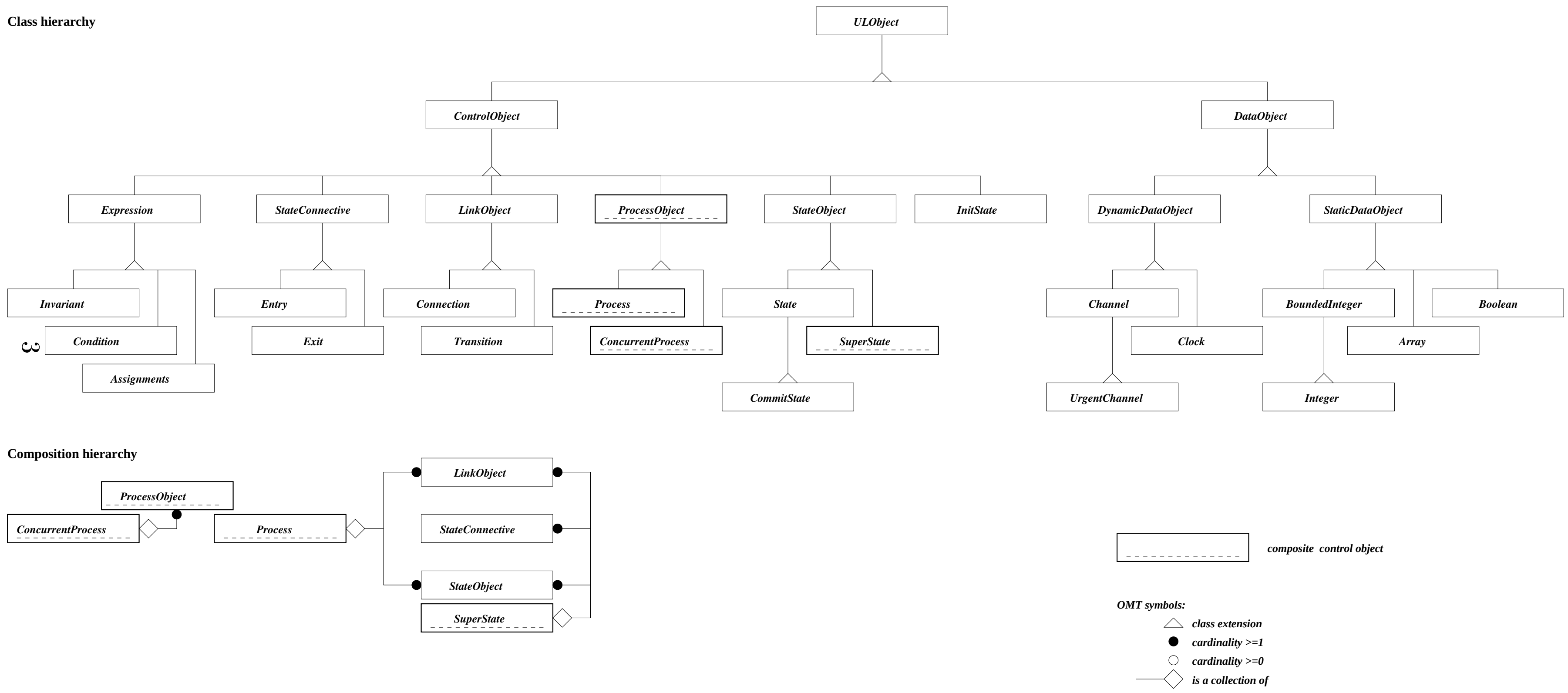

Figure 2: UL++ class hierarchy.