Towards Type Systems for Dynamic Components (Abstract)

Arnd Poetzsch-Heffter

FernUniversitäat Hagen

In an object-oriented setting, a system is a collection of communicating objects. Current programming techniques do not support mechanisms to structure such collections, e.g. by providing constructs to group objects and to enforce interfaces of such groups. In the following, we call a group of objects that work closely together a *(dynamic) component*. We assume that two components are either disjoint w.r.t. the objects they contain or that one component is part of the other component, i.e. all objects of the former component belong as well to the latter. Consequently, components induce a hierarchical structure on the objects in a system.

The component structure introduces a boundary between objects inside and outside the component. In particular, given a component C, we can distinguish four kinds of objects:

- internal: objects in C that must not be referenced from the outside;
- interface: objects in C that may be referenced from the outside are called interface objects;
- used: objects outside C that are referenced from objects in C are called used objects;
- external: objects not directly referenced from objects in C.

All the communication of a component with the outside objects can be controlled by the interface and used objects. The types of the interface objects describe what kind of services the component provides; as in all typed OO-languages, the types determine which methods can be invoked on interface objects, i.e. from the outside. The types of the used objects explain what kind of services the component needs from the outside. Based on these types, the behavior of a component can be characterized. In addition to the types, the number of interface and used objects and possibly some information about their relationship might be of interest. In general, the number and relationship of interface and used objects can change dynamically.

In this talk, we present an approach for typing the interfaces of dynamic components. The basic idea is to use regular expressions to control the dynamic change of the number and relationship of interface and used objects. The alphabet of such a regular expression R is given by $\{\text{interface}, \text{used}\} \times$ TYPES where TYPES is the set of reference types of the underlying object-oriented program. If $w_1 \ldots w_n$ is a word in LANG(R), then the component may have n interface and used objects where the kind and type of the i-th reference is constrained by the letter w_i. Relationship information is captured by the derivation trees according to R.