Behavior Consistent Refinement of Object Life Cycles

Michael Schrefl¹ and Markus Stumptner²

¹ Institut für Wirtschaftsinformatik, Universität Linz, Altenbergerstr.69 A-4040 Linz, Austria. email: schrefl@dke.uni-linz.ac.at
² Institut für Informationssysteme, Technische Universität Wien, Paniglg. 16, A-1040 Wien, Austria. email: mst@dbai.tuwien.ac.at

Abstract. This paper examines the inheritance of object life cycles that are specified by behavior diagrams. A behavior diagram of an object type models possible life cycles of its instances by states, activities and arcs corresponding to places, transitions and arcs of Petri Nets. In an inheritance hierarchy, subtypes specialize the life cycle of supertypes by extension, i.e., adding states and activities, and (the focus of this paper) refinement, i.e., decomposing states and activities into substates and subactivities.

The main contribution of this paper is the identification of necessary and sufficient rules for checking behavior consistency between a behavior diagram of a type and a refined subtype, as well as for the combination of extension and refinement.

Keywords: object-oriented design, object life cycle, inheritance

1 Introduction

One of the central aspects of object-oriented systems is inheritance. Inheritance defines a relationship between two object types, where one object type called the subtype inherits structure and behavior from the other one called the supertype. The inherited structure and behavior may be further specialized in the subtype. Subtypes may extend supertypes by new features or redefine inherited ones. Applying inheritance to features requires a clear understanding of what it is that is actually inherited. In particular, it is common to distinguish between individual operations and more comprehensive properties. Several object-oriented design methods, such as OMT [10], OOSA [5], and OBD [7] model the behavior of object types at two interrelated levels of detail: at the activity level and at the object type level. For example, behavior diagrams [7] specify at the activity level the signature of an activity by identifying types and preconditions of input parameters as well as the type and the postcondition of the return value. At the object type level, object behavior is specified in terms of object life cycles that identify legal sequences of states and activities.
Inheritance of activities corresponds to inheritance of operations in programming languages, which is fairly well understood [3, 15]. Recently, more attention has been directed towards examining inheritance of object-life cycles, i.e., how the life cycle of a subtype should relate to the life cycle of its supertype. We have been studying inheritance of object life cycles in the realm of behavior diagrams [6, 7, 12, 1] which are based on Petri nets. A behavior diagram of an object type represents the possible life cycles of its instances by activities, states, and arcs corresponding to transitions, places, and arcs of Petri nets. Subtypes may specialize the behavior diagram of supertypes in two ways: by extension and refinement. Extension means adding activities, states, and arcs. Refinement means expanding inherited activities and states into subdiagrams. Activities of behavior diagrams can be further described by activity specification diagrams, activity realization diagrams, and activity scripts. The additional diagram types are not needed in this paper; see [1, 7] for details.

In [13], we have already presented a set of complete rules for checking the consistency of lifecycles for objects whose behavior diagrams are created by extension of the behavior diagrams of a supertype. This work was driven by the principle that a subtype should either preserve the "observable behavior" or the "invocable behavior" of its supertype, as classified by Ebert and Engels [4]. The notion of observable behavior is given by the assumption that if one ignores the parts of the lifecycle specific to the subtype, the lifecycle of an instance of the subtype should appear just as a lifecycle of an instance of the supertype. The notion of invocable behavior corresponds to the assumption that any sequence of operations that is invocable on a supertype instance should also be invocable on a subtype instance.

This paper deals with the definition of consistency requirements for the refinement of behavior diagrams. Our initial work on refinement [12] introduced a set of structured refinement primitives. In this paper we take a more general approach based on labeling states and activities in the diagram. The idea of labeling states and activities is inspired by the way in which in business processes guided by paper are executed: different copies of a business form are given different colors. Each business activity handles one or several copies of a form, it may collect several copies of a form from different input boxes, and it may distribute several copies of a form to different output boxes. In this analogy, labels in behavior diagrams correspond to the colors given to different copies of the form. Note that, if an activity has been refined, its original description in the supertype is "abstract" (which is why it had to be refined) and it cannot be invoked on an object (cf. [12]). Therefore the notion of invocation consistency is not applicable to refined subtypes. This paper deals exclusively with guaranteeing observation consistency.

Many papers have been published on behavior and equivalence preserving refinements of Petri nets (cf. [2] for a survey), but as it turned out, behavior consistency for refinement is much easier to check for labeled behavior diagrams than for Petri nets in general.