Synthesizing Design Models from Scenarios by Learning

Benedikt Bollig[1)], Joost–Pieter Katoen[2)], Carsten Kern[2)], Martin Leucker[3)]

1) *ENS Cachan*
Laboratoire Spécification et Vérification
94235 Cachan Cedex, France
bollig@lsv.ens-cachan.fr

2) *RWTH Aachen University*
Software Modeling and Verification Group
52074 Aachen, Germany
{katoen,kern}@cs.rwth-aachen.de

3) *TU Munich*
Institut für Informatik
85748 Garching, Germany
leucker@in.tum.de

Abstract:

The elicitation of requirements is the main initial phase in the typical software engineering development cycle. A plethora of elicitation techniques for requirement engineering exist. Popular requirement engineering methods, such as the Inquiry Cycle and CREWS [→Nuseibeh'00], exploit use cases and scenarios to specify the system's requirements. Sequence diagrams are also at the heart of the UML. A scenario is a partial fragment of the system's behavior, describing the system components, their message exchange and concurrency. Their intuitive yet formal nature has resulted in a broad acceptance. Scenarios can be positive or negative, indicating a desired or unwanted system behavior, respectively. Different scenarios together form a more complete description of the system behavior.

The following design phase in software engineering is a major challenge as it is concerned with a paradigm shift between the *requirement* specification—a partial, overlapping and possibly inconsistent description of the system's behavior—and a conforming *design model*, a complete behavioral description of the system (at a high level of abstraction). During the synthesis of design models, usually automata-based models that are focused on intra-agent communication, conflicting requirements will be detected and need to be resolved. Typical resulting changes to requirements specifications include adding or deleting scenarios, and fixing errors that are found by a thorough analysis (e.g., model checking) of the design model. Obtaining a complete and consistent set of requirements together with a related design model is thus a highly iterative process.

We propose a novel technique that is aimed to be an important stepping stone towards bridging the gap between scenario-based requirement specifications and design models. The novel aspect of our approach is to exploit *learning* algorithms for the synthesis of *distributed* design models from scenario-based specifications. Since message-passing automata (MPA, for short) [→Brand'83] are a commonly used model to realize the behavior as described by scenarios, we adopt MPA as design model. We present a procedure that interactively infers an MPA from a given set of positive and negative scenarios of the system's behavior provided as message sequence charts (MSCs). This is achieved by generalizing Angluin's learning algorithm for deterministic finite-state automata (DFA) [→Angluin'87] towards specific classes of bounded MPA, i.e., MPA that can be used to realize MSCs with channels of finite capacity. Moreover we investigated classes of regular MSC languages and corresponding MPA which can (not) be learned.