SystemC [89] is increasingly being used for system level modeling and simulation during the first phases in a product’s design cycle. The modeling and simulation undergoes ample validation and debugging before resulting in a functionally correct SystemC implementation model. Validation by simulation requires designers to generate a set of input sequences that exercise all the relevant and important properties of the system.

A property of the system describes a characteristic of that system. For example, a FIFO component may ignore inputs when it is full. Testing this property requires the input sequences that transition the system to that state. Thus, one needs to generate input sequences which cover trivial and nontrivial cases of a system. However, coming up with these input sequences are not always trivial. This is particularly true for large and complex designs where it is difficult to identify and describe the input sequences to reach a particular state [18, 66]. Furthermore, there may be multiple paths (different input sequences) that transition the system to the same state. Authors in [18] provide test case generation by performing static analysis on the SystemC source and then use supporting tools for generating tests. This approach is often limited by the strength of the static analysis tools and the lack of flexibility in describing the reachable state or states of interest for directed test generation. Also, static analysis requires sophisticated syntactic analysis and the construction of a semantic model, which for a language like SystemC built on C++, is difficult due to the lack of a formal semantics for it. In fact, [18, 66] does not precisely describe a semantics for SystemC. It is also difficult to diagnose the exact transition that causes a failed test case execution. For this reason, it is important to provide designers with a methodology and set of tools that ease the burden of directed test case generation and diagnosis.

SpecExplorer is a tool developed by Microsoft for model-based specification with support for conformance testing. The essence of SpecExplorer is in describing the system under investigation as a model program either in AsmL [115] or Spec# [77] and performing conformance tests on an implementation model in the form of some software implementation. The model
program in AsmL serves as a formal specification of the intended system because AsmL employs the Abstract State Machine (ASM) [15, 45] formalism. From here on, we use “specification” interchangeably with “model program”. ASMs allow for specifying hardware or software systems at the desired level of abstraction by which the specification can focus on only the crucial aspects of the intended system. The added quality of ASM specifications being executable, aids in verifying whether the specification satisfies the requirements, whether the implementation satisfies the specification and transitively whether the implementation satisfies the requirements. This makes ASMs and SpecExplorer a suitable formalism and tool respectively for semantic modeling, simulation and as we show in this paper, for exploration and test case generation (Figure 7.1).

Previous works in [48, 47, 37] use SpecExplorer to put forward a development and verification methodology for SystemC. The difference is that they focus on the assertion based verification of SystemC designs using Property Specification Language (PSL). Their work mentions test case generation as a possibility but this important aspect of validation was largely ignored. We were not able to employ any of the work done by these authors because their tools are unavailable. Their formal discrete-event specification in AsmL, translators from PSL to AsmL, and AsmL to SystemC were not available even after several requests.

In this work, we present a model-driven method for not only specifying and developing but also validating system level designs for SystemC. We do not address the full validation of system level designs, but a small subset. In particular, we use our validation method to test corner case scenarios for which manually coming up with input sequences for testbenches may be very difficult. Figure 7.2 shows a block level schematic of the model-driven methodology. We create an abstract model from a natural language specification and this abstract model is what we call the semantic model. It is specified using AsmL (an implementation language for ASMs). Since, SystemC follows the