

# Search-based Execution-Time Analysis in Component-Oriented Real-Time Application Development

Hans-Gerhard Gross and Nikolas Mayer

*Fraunhofer Institute for Experimental Software Engineering  
67661 Kaiserslautern, Germany  
{grossh,mayern}@iese.fhg.de*

## 1. Introduction

One of the most important motivations for the application of object technology and subsequently component-based software engineering techniques in practice is that new applications can be created with significantly less effort than in traditional approaches, simply by assembling the appropriate prefabricated parts. However, contemporary object and component technologies are still some way from realizing this vision, especially when component-based real-time applications are considered. The late integration implied by the assembly of deployable objects or components means there is little opportunity to verify the correct functional and non-functional operation of the resultant application before deployment in the run-time environment. This problem is compounded when non-functional requirements are considered, for example the compliance of the application to a real-time schedule. Such real-time requirements are not only affected by individual objects, but by the entirety of all objects that make up the application. Timing verification can only be performed when components are assembled and put together into a new configuration. Although, the effort involved in plugging components together may be relatively small, therefore, the effort involved in verifying that the resulting assembly of components works as expected, and shows the expected run-time behaviour, may be much greater. The savings that are promised by object and component technologies may thus be wiped out by the extra effort needed at integration and deployment time to verify that the run-time behaviour of the resultant application is acceptable.

## 2. Dynamic Timing Analysis

Dynamic timing analysis can be defined as software testing with the violation of the timing schedule as test criterion. This in fact, represents a typical search/optimisation problem that can be tackled by a search/optimisation method with the actual test cases representing the optimisation parameters. Each parameter set is a vector that defines an input scenario for the task under test. The cost function is deter-

mined by the time it takes to execute the task with a given input combination. Advanced search algorithms such as hill-climbing or tabu-search realize optimisation that is guided through the fitness function. Evolutionary algorithms are also belonging to this class, and they have been found most effective for timing validation purposes [5]. This applies genetic algorithms [2] or evolution strategies [4] to typical software testing problems [6]. The target is to find tests that represent the longest or shortest execution time of a program, or to violate its schedule [3,6]. However, the technique has not yet been applied to timing analysis of object-oriented and component-based real-time systems. The essential difference between the procedural paradigm and the more recent object-oriented development paradigm lies in how data and functionality are treated. Whereas the first propagates a strict separation of data and functionality, the second encourages the exact opposite, the combination and encapsulation of data and functionality. All internal state variables are by definition hidden to outside entities of an object. This also includes the test software, in this case represented by the search technique. States can only be accessed and changed through the provided functional interface of the class. In general, only a distinct history of interface operation invocations will define the combination of the internal attributes that makes up an internal state. The execution time is fundamentally dependent upon an object's internal state information, and therefore the state information must be part of the optimisation process.

## 3. Search-Based Testing for Components

Search based execution-time analysis that is applied to object-oriented and component-based real-time systems is relying on the optimisation of input parameters according to the method invocation history plus a specific architecture. The organisation of the test software (evolutionary algorithm) draws its concepts from the built-in contract testing technology developed within the EC *Component+* project [1]. Contract testing has been specifically developed in order to check the contract compliance of pair wise interacting components at integration

and configuration time. In a real-time system, a contract between two components additionally comprises timing properties that a server component is expected to fulfil in order to satisfy the contract of its client. Contract testing augments components with built-in test software and test interfaces that implement an automatic checking mechanism for component interactions.

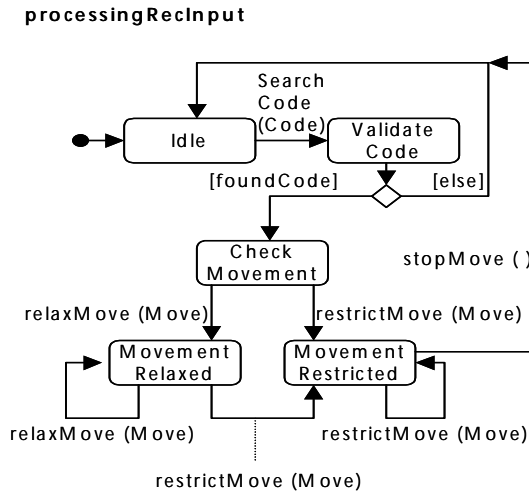


Figure 1. Behavioural model of a simple door control unit

The invocation history of an operation call depends upon the states through which execution must proceed in order to be able to perform an operation. For example Figure 1 displays a behavioural model of a simple vehicle door control unit. In order to be able to execute and assess the timing of *relaxMove*, the operation *searchCode* must be invoked. This takes a parameter *Code* that may have an effect on the execution time of *relaxMove*. The same is true for the operation *restrictMove*. This may show different timing if it is invoked in the state *MovementRestricted* or in the state *MovementRelaxed*. For search-based timing analysis it means that we have to optimise not only the input parameter set for the operation under consideration, but also all input parameter sets for operations that have been executed prior to the subject since these determine the unit's current state. Table 1 shows an example invocation paths plus the input parameters that must be considered in optimisation.

Table 1. Feasible invocation path for assessing the timing of operation *restrictMove*

Operation History	Optimised Parameters	Final State
SearchCode	Code	CheckMovement
RelaxMove	Move	MovementRelaxed
RelaxMove	Move	MovementRelaxed
<b>RestrictMove</b>	<b>Move</b>	<b>MovementRestricted</b>

A test case for this example comprises an invocation history that must be executed in order to set the required internal state of the tested object, and the optimisation of the four input values that these invocations require. This collectively defines a test that is incorporated together with the optimisation process into a client of the component. Such an augmented client can then assess the timing compliance of its server whenever the two are plugged together and integrated in a new application.

#### 4. Conclusion

The work presented in this paper creates the basis for using search-based testing technology in component-oriented software construction. We have introduced a methodology for applying evolutionary testing techniques to timing analysis of object-oriented (and consequently component-based) real-time systems. This is based on optimising the invocation history of an operation additionally to the actual parameter of the method under timing test. We believe that only through the introduction of such suitable and powerful verification mechanisms can the full vision of object technology and component-based development become a reality in real-time system construction.

#### References

- [1] Component+, Technical Report, Built-In Testing for Component-Based Development. EC IST 5<sup>th</sup> Framework Programme IST-1999-20162 (www.component-plus.org), November 2001.
- [2] Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, 1989.
- [3] Müller & Wegener. Comparison of static analysis and evolutionary testing for the verification of timing constraints. RTAS'98, Denver 1998.
- [4] Schwefel & Männer. Parallel Problem Solving from Nature. Springer, 1990.
- [5] Tracey. A search-based automated test-data generation framework for safety-critical software. PhD Thesis, University of York, 2000.
- [6] Wegener & Grochtmann. Verifying timing constraints of real-time systems by means of evolutionary testing. Real-Time Systems 3(15).