1. (a) **Project title:** Systematic Testing of Realtime Embedded Software Systems
   (b) **Project Acronym:** STRESS
   (c) **Principal Investigator: prof dr H. Brinksma (UT-INF)**

2. **Summary:**

The aim of this project is to develop the theory and tools for advanced model-based test generation and execution of realtime embedded software systems. It seeks to extend the capabilities of the existing TorX testing tool, a successful testing tool for non-realtime features of reactive systems. The work will address two main extensions, viz. test generation and execution for realtime features, and test data selection for system parameters with large domains. On the theoretical level this requires the integration of the theory of suspension trace refusals with timed automata, and the combination of classical data selection techniques from software testing (e.g. equivalence partitioning) with realtime concurrency theory. On the level of tools the project seeks to combine TorX technology with that of Uppaal.

3. **Classification:**

Computer Science NVI Classification: 3.2, 3.4, 6.3, 6.4, 6.5
NOAG-i 2001-2005: embedded systems, software engineering, algorithms and formal methods

4. **Composition of Research Team.**

Name	Research Field	Affiliation	hours/week
Prof. dr. H. Brinksma	Formal Methods Tools	Fac. Comp. Sci. Un. Twente	4
Prof. dr. K.G. Larsen	Formal Methods Tools	Fac. Comp. Sci. Un. Twente/Aalborg Un.	4
Dr. ir. G.J. Tretmans	Software Technology	Comp. Sci. Inst. Un. Nijmegen	4
Prof. dr. ir. M.J. Plasmeijer	Software Technology	Comp. Sci. Inst. Un. Nijmegen	2
Dr. B. Nielsen	Distributed Systems	Comp. Sci. Inst. Aalborg Un.	p.m.
AIO (vacancy)		Fac. Comp. Sci. Un. Twente	36
AIO (vacancy)		Comp. Sci. Inst. Un. Nijmegen	36
programmer (vacancy)		Fac. Comp. Sci. Un. Twente	36

Prof.dr. K.G.Larsen holds a part-time professorship at the University of Twente, and is a full professor at the Distributed Systems and Semantics Group of the Computer Science Department of Aalborg University, Denmark. There exists a strong scientific cooperation between his group in Aalborg and the Formal Methods Group in Twente in the area of validation of realtime distributed systems, also in the frame of the EU IST projects AMETIST and ARTIST. Dr. B. Nielsen is a member of the Aalborg team who will be involved especially in the cooperative research on realtime testing

At the time of writing Dr.G.J.Tretmans still holds a research position at the formal methods group at Twente. Starting April 1st, 2002, he will assume a position as an associate professor at the Software Technology group of the University of Nijmegen.

5. **Research School.** Both the *Formal Methods and Tools* group at Twente, and the Software Technology group at Nijmegen participate in the Research School of the Institute for Programming Research and Algorithmics (IPA).

6. **Description of Proposed Research.**

   (a) **Problem statement and objectives.**

   In the past decade the development of theories and software tools to support the testing of hardware and software systems has become an academically respectable area of research. Whereas testing was once condemned as an inferior validation method – *testing can only show the presence of errors, not their absence* – it is now recognized as a very important technique of quality control, whose flexibility and scalability make it the most applied validation technique in practice. In computer science research it has been identified as a promising application area for formal methods, where semantic models and theories have been usefully applied to develop formal testing theories. These have been used to give precise interpretations of fundamental testing concepts, such as correctness of implementations with respect to specifications, soundness and completeness of test suites, test coverage, etc. Based on such formal definitions algorithms can be developed for the generation of tests from formal specifications, the validation of tests, and the interpretation of test results, among others. As most practical testing is performed on an ad hoc basis, and with a low degree of automation, such results are not only theoretically interesting, but also have enormous practical potential. This should be understood in the current industrial setting, where testing may take up 30 to 50% of the budget of a software development project.

   Concrete examples of formal methods based test environments are TVEDA [Pha94], TGV [JM99], and TorX [BFV+99]. TVEDA and TGV are tools that have been developed by researchers from France Telecom and IRISA, in France. TorX has been developed at the University of Twente, in collaboration with Eindhoven University, Philips Research, KPN Research, and Lucent as part of the STW Côte-de-Resyste project [STW96]. Its main features are:

   - automated test generation for functional conformance properties from formal specifications. A distinguishing feature of TorX is its solid theoretical foundations, with its test derivation algorithms firmly rooted in a specialization of De Nicola/Hennessy-testing theory to input/output-automata, *quiescent refusal testing* [DNH84, Tre96b].
   - support for test execution. TorX supports both *on-the-fly* test execution, i.e. test execution is directly driven by the generated test steps while they are being calculated, and *batch* test execution, i.e. test execution is based on test suites that have been generated before and stored.
   - an open, modular architecture. This lends the tool great flexibility in handling different specification formalisms. In principle any formalism whose specifications can be interpreted as labelled transition systems [Mil89] can be used in combination with TorX, if an appropriate front-end translation module is provided. Currently, such translators are available for the formalisms LOTOS [ISO89] and Promela [Hol91]. At the back-end, an application-dependent adapter module makes it possible to cater for a wide variety of test methods [ISO91] and architectures for the SUT (system under test).

   On the basis of a number of realistic applications (the EasyLink protocol [BFHV01], 'Rekeningrijden' [VBF02], Cell Broadcast Centre [Chr01]), it has become clear that TorX is a practically useful conformance test tool that compares favourably to other existing approaches [HFT00]. At the same time, two major areas for improvement have been identified, viz.:

   - *the lack of support for dealing with realtime features in specifications and implementations.* TorX and its underlying theory are restricted to untimed behaviour, and indeed many, and probably most, aspects of a system can be validated in an untimed setting. Still, most embedded systems do have features that depend upon their timing in an essential way,

which are error-prone and must therefore be subjected to systematic validation methods, such as testing.

Even if the functionality of a system itself is not time-dependent, its implementation may have realtime features that interfere with testing. This typically has to do with the use of timers that are used to guarantee the progress of a system. The time-out values determine the observational criteria for identifying the deadlock/livelock properties of the SUT. This problem occurred in the 'Rekeningrijden' application [VBF02].

- *the lack of support for dealing with realistic data domains.* TorX and its underlying theory mainly deal with the control aspects of system behaviour. They originate in the analysis of protocols in communication networks and distributed embedded systems, in which the flow of control is the determining factor in system correctness. These control aspects are usually modelled by some kind of state machine or labelled transition system. The relevant data structures and domains are restricted to simple types (e.g. bits, bytes, subranges).

  With the advent of embedded (and communication) systems whose functionality is mainly software-dependent, this situation has changed. The correctness of these systems is also related to the correctness of complex data structures, variable values ranging over large or infinite domains and constraints involving these values.

  Another reason for dealing with data-intensive applications is the natural trend toward wide-spectrum (testing) tools. Users have a preference for tools that can be applied generically, so that working on different application areas does not require using different tools and/or methods. This tendency makes it desirable that tools like TorX can also be applied in the setting of more traditional, data-intensive software testing.

  Currently, the support for data testing, both in theory and in tools, is insufficient as the experiences with EasyLink [BFHV01] and the Cell-Broadcast-Centre [Chr01] have shown.

A success story in the application of formal methods to industrial problems is system verification by model checking. In the past decade a substantial number of model checking tools have been developed and successfully applied, such as SPIN, SMV, Murϕ, JAVApathfinder, etc. A number of such tools have also been developed for the analysis of real-time systems such as Kronos, Hytech and Uppaal [Sof97].

Uppaal is a tool for modelling, simulating, and verifying real-time systems, developed in a collaboration between the Department of Computer Systems at Uppsala University and the Department of Computer Science at Aalborg University. The tool allows for the modelling of real-time systems as collections of non-deterministic processes with finite control structure and real-valued clocks (so-called timed automata) communicating through channels and shared variables.

Much of Uppaal related research is devoted to new data structures and algorithms for efficient symbolic representation and manipulation of the state spaces encountered during exploration of system models. Through its lifetime, the performance of the tool has improved substantially: on the full collection of industrial case studies both space-and time-performance has improved by a factor of 10 every nine months. In particular, symbolic data structures dealing with the infinite, continuous part of the state space due to the presence of clocks have been a primary concern. One such data structure is a *zone* which allows certain subsets of the Euclidean space to be represented as Difference Bound Matrices that constitute the central data structure in the Uppaal verification engine.

A first attempt to apply the efficient symbolic techniques of Uppaal to the problem of test generation based on timed automata specifications has been carried out in Nielsen's dissertation [Nie00]. The results have been implemented in the RTCAT test case generation tool. Applications to realistic cases show promising results. They also identified a number of open issues, such as the limited expressiveness of the specification formalism, the lack of a solid testing theory for dense timed systems, limited size of the systems that can be systematically analysed, and the efficient handling of timing non-determinism (timing uncertainty) in specifications, implementations, and test execution environments.

From the overview above it is clear that the objectives of the STRESS project should be:

(1) *development of a realtime theory of quiescent refusal testing.* In order not to sacrifice the formally robust approach of TorX, the effort to support the testing of realtime system features must be based on solid theory. In our case this entails working on a realtime extension of the theory of quiescent refusal testing.

(2) *development of a testing framework for complex data structures.*
As under the previous point, we must also extend the theory of quiescent refusal testing with ways to deal with data-oriented aspects. This involves methods to specify, represent and manipulate complex data structures and large data domains in such a way that the extension is well-defined, consistent with the theory of quiescent refusal testing, and leads to computationally tractable algorithms for test generation and test data selection.

(3) *implementation of a test tool environment for embedded software systems.*
The goal is to extend and adapt TorX with features that support the two previous objectives: realtime testing and testing of data-oriented aspects. The tool should be able to generate tests from specifications in the appropriate formalisms, execute these tests, and analyse their results. It will be most challenging to create a realtime test execution environment that both precise and sufficiently fast to support realtime interaction with the system under test. This will require a careful revision of the existing tool architecture, and of the TorX and Uppaal components that we intend to reuse. Their use ensures the availability of robust and proven technology for untimed, control-oriented testing, exploits the considerable investments already made in the tools and should make compatibility with the current version of TorX easier to achieve.

(b) **Method**

To explain our approach to the realization of the objectives we start with three sections that give background information relevant to the scientific ingredients of the proposed project. They are followed by sections describing the way in which the actual research is to be carried out.

Quiescent refusal testing. The testing theory underlying TorX has its roots in the seminal work of De Nicola and Hennessy that proposed a family of semantic models for process algebras and labelled transition systems based on observing of the behaviour of processes by composing them in parallel with observer processes [DNH84, DN87]. Such so-called testing pre-order models studied as semantic models for reactive systems. Their first use in the context of actual testing was proposed by Brinksma [Bri88], where the theory was used to derive from a specification a set of observers that suffices to detect non-conforming implementations.

Subsequent work by Tretmans improved on this basic testing theory in at least two respects. First, it replaced the assumption that SUT and its tester (observer) are communicating via synchronous interaction [Tre96a]. In practice, testing almost always takes place in a context where communication is asynchronous. This aspect was taken into account by specializing the testing theory to the class of input/output automata [LT89]. In this class asynchrony is implicitly modelled by the assumption of *input-enabledness*, i.e. the assumption that in each state the system has a defined reaction to input from its environment. In asynchronous communication this property is typically enforced by the buffer capacity of the communication channels between processes. In this setting a process is never in a deadlocked state as it can always accept new inputs. The relevant concept here becomes *quiescence* [Vaa91], i.e. the phenomenon that the process refuses to produce any output if no further input is provided.

A second improvement by Tretmans was the generalization of I/O-testing to *repetitive quiescence*, i.e. the phenomenon that after observation of quiescence testing may continue by providing new input of the system [Tre96b]. In practice, quiescence is assumed to have been observed after the elapse of some system dependent maximal response time. In order not to waste information it is attractive to try other test sequences out of a quiescent state when it is reached. This idea is a generalization of the work on *refusal testing* by Phillips [Phi87] (and,

independently, Langerak [Lan90]), in which testing continues after the observation of a refusal (the incapacity to synchronize on a given set of actions).

Testing timed automata. Test derivation from timed automata is still a fairly undeveloped research area. Although some work has been done [SVD97, ENDK98, CO00, CKL98, SS01] (see section on related work), they all suffer from more or less serious defects, such as state space explosion problems, or problems with the soundness and completeness of the derived test suites.

So far only the work by Nielsen has led to results with a potential for practical applicability [Nie00, NS00, NS01a, NS01b, NS02]. The main features of his work, as implemented in the RTCAT tool, are

- test generation is based on a restricted class of dense and potentially non-deterministic, timed automata specifications. Also here, the work makes use of (a simple timed extension of) De Nicola and Hennessy's testing pre-order theory [NH84].
- test selection for realtime systems (choosing at which time instances inputs should be supplied to the system under test) is addressed by partitioning the state space into coarse grained equivalence classes that in a systematic way preserve essential timing and deadlock information, and selecting a few instances for each class. This approach is inspired by sequential black-box testing techniques frequently referred to as domain- or partition testing [Bei90], by regarding the clocks of a timed specification as (oddly behaving) input parameters.
- the test case generation algorithm systematically ensures that the specification will be covered such that the relevant Hennessy tests for each reachable equivalence class are generated. UPPAAL verification techniques are used to interpret the timed automata specification, to compute and cover the reachable equivalence classes, and to compute the timed test sequences.

Data testing. Testing and test derivation for data-oriented aspects based on formal specifications is still an underdeveloped area. The best developed theory, to our knowledge, the algebraic abstract data type testing theory by Gaudel et al. [Gau95]. Test data are selected based on formally expressed assumptions referred to as test hypotheses. A tool is available [Mar95]. Also some approaches to testing based on Z specifications are known [Hie97], most of which are based on some kind of formalization of the classical equivalence partitioning approach [Mye79]; see also the section on related work.

The current approach toward dealing with data in TORX is inherited from the front-end translation tools which are used. The approach is rather naive and straightforward: all data domains are expanded by explicit enumeration of all their possible values. This leads to models where all data aspects are completely encoded in the control flow (labelled transition systems), which leads to obvious disadvantages: data domains can only be of limited size, combining multiple data variables leads to the infamous state-space explosion problem, and important information on how data and data sets are structured is lost. Such structuring information is useful for test data selection.

Developing a theory of realtime testing A robust theory that can serve as the foundation for a TORX-UPPAAL realtime testing tool must provide a suitable formal implementation relation that characterizes precisely when an implementation conforms to a given specification. Such a relation also provides indirectly the formal criterion of what should be tested, and serves as the basis for test generation.

It stands to reason that in the STRESS project it should be attempted to apply the ideas of quiescence refusal testing to timed automata. To do so the following technical issues must be addressed:

- *input-enabledness in timed automata.* In the untimed case input-enabledness is an abstract semantic representation of the fact that in practice SUTs (systems under test) cannot refuse to accept new (test) inputs because of the buffer capacity of the test environment of an implementation. They are usually chosen to be unbounded, which presents a problem in the case of timed automata, whose control state space must be finite. It will important to identify the right class of input-enabled timed automata that suffice to model the SUTs and have finite control. Fortunately, realtime media of a test system typically have a finite maximal transfer delay, which suggests that bounded media suffice. An elaboration of quiescent refusal testing for bounded I/O capacity can be found in [Hee98].
- *time delay actions and quiescence.* Observations of timed automata do not only involve observable actions, but also observations of the passage of time, usually formalized by special *time delay actions.* In the presence of such actions the notion of quiescence must be refined. The refusal of a time delay corresponds intuitively to (bounded) non-quiescence: the system must react before such a delay would have occurred. A system is quiescent if can only execute a sequence of diverging delays.
- *nondeterminism in timed I/O automata.* Testing typically occurs in a context of nondeterminism. Even if an implementation itself is deterministic the test context (the environment needed to execute the test) will usually introduce nondeterminism into the system. Nondeterministic timed automata are strictly more expressive than deterministic ones, and language inclusion between nondeterministic timed automata is undecidable [AFH94, YJ94]. This is not a death-blow to testing theory, as complete testing is only obtained as a theoretical limit of finite approximations, but it is clear that a judicious choice of the allowable class of nondeterministic systems must be made to obtain good (practical) results.

Developing a theory of data testing. New methods of dealing with complex data structures and large data domains in testing have to be developed. Whereas for realtime testing there is an obvious candidate formalism – timed automata – and tool – UPPAAL– for integration with quiescent refusal testing and TorX, this is not the case for integration of data-oriented aspects. Consequently, the research consists of two phases. Firstly, criteria have to be identified and analysed for selecting a data formalism with accompanying tool support and, secondly, data test theory has to be elaborated for a selected data formalism.

Since it is unlikely that one particular data formalism will suit all possible application areas of data testing, the results of the first phase are at least as important as of the second. A modular approach must be developed where criteria are given for a well-defined interface between the control-flow formalism and the data formalism to allow different data formalisms to be plugged in. Such an interface definition should extend from the semantic level to the level of a modular tool structure. In this respect it is noted that the interface should be flexible enough so that not only semantically well-defined (academic) languages can be considered, but also standard (industrial) languages for data description as IDL [Nes90], ASN.1 [ISO90] and XML [GQ99].

For a data formalism at least the following aspects have to be considered. First, it should consistently integrate and not sacrifice or obscure the existing well-defined theory of quiescent refusal testing. Second, an important issue is finite and efficient representability of data. Instead of explicit enumeration of all possible values, other, more symbolic ways of representing data and data domains are necessary. Thirdly, the manipulations and calculations which are necessary for test generation and test data selection have to be taken into account. It should lead to computationally tractable, implementable and efficient algorithms and/or heuristics for test generation and test data selection. Fourthly, usability of the data formalism should be considered or, at least, it should allow a user-friendly data specification language to be built on top of it. Finally, since it is not the prime intention of the STRESS project to define yet another data specification formalism, selecting, adapting and integrating an existing data formalism is preferred. In this respect is important to consider also existing tool support and the feasibility of consistent and modular integration within the TorX architecture.

Considering the above constraints there are many candidate data formalisms and accompanying tools, which may be considered. A prime candidate is the use of functional languages. Functional languages enjoy a rich underlying theory, they allow to define (infinite) data structures and computations on these structures are easily specified. Moreover, elaborate and professional tool sets are easily available including static analysis and typing, interpretation, compilation, rewriting and dedicated proof support. In particular, the project has easy access to the Clean System which is an environment for the functional language Clean [PE93, PE99, PE01]. It includes the theorem prover Sparkle [MEP02]. Provided it meets the criteria for integration into the formal testing framework, the combination of Clean and TorX will be elaborated. It is expected that the computations necessary for test generation can be expressed over functional languages. It might be that for test data selection extensions are necessary, e.g., incorporating ideas from constraint solving techniques.

Alternative approaches exist, should the approach via functional languages turn out to be less successful. Other candidate data formalisms include algebraic abstract data types as currently used within LOTOS, or some (user-friendly) extensions thereof, with corresponding rewriting and narrowing tool support. In this respect the data selection tool LOFT which works on such data types can be useful [Mar95]. Also languages which are supported by powerful theorem provers, e.g. PVS [COR+95], may be considered. This is triggered by our conjecture that for the computation of quiescence in a symbolic setting some kind of proof support will be necessary. But also more concrete data type languages derived from programming languages can be considered, just as industrial languages for the description of data, like IDL or ASN.1; see above.

Common Issues. The common issue in both timed testing and data-oriented testing is that in the current test theory (and TorX) they lead to large or infinite state systems. Exploring finite and efficient representations for such state spaces, lifting implementation relations and test algorithms to these representations and transforming the results of the algorithms into executable tests are problems which are encountered for both in an analogous manner. Considerable effort will therefore be given to finding generic solutions that can be instantiated for both problem areas.

One particular common problem is that of *test selection*. If transition systems are large or infinite exhaustive testing is not possible. The question then is how to select the data values or the instances in time for a test case. The principle observation here is that there can be no formal argument why one instance or data value is better than another. Selection is usually based on heuristic arguments. Examples of such arguments can be found in classical software testing, e.g. equivalence partitioning or domain testing (partitioning the input space into subsets based on the assumption that all values in a subset have an equal chance of being right or wrong; this technique was implemented in RTCAT; see above) or boundary value analysis (based on the assumption that values on the boundary of input partitions have a larger chance of leading to erroneous behaviour). Although based on heuristics and not on formal arguments, there is a wish to measure the error detecting capability of test suites generated with different strategies. Such measures, referred to as coverage functions, can be used to compare test suites, to select the best ones, and, indirectly, to compare system implementations passing such test suites. First attempts have been made in this direction [Bri93, MV95, CG97, FGST02]. Test selection strategies and their specifications, the integration of (classical) selection heuristics in the formal framework, and measures for expressing coverage and error-detection capabilities of a test suite will be investigated within the STRESS project.

Another issue of common interest are application case studies to test the theory and the tools. The case studies will be selected later in the project (year 3) on the basis of their suitability to evaluate the progress made in realtime test generation, realtime test execution, test data selection, test selection, etc. Both cases combining these aspects, and those focussing on specific aspects will be relevant. The provision of these case studies, as well as smaller example cases for initial studies, is warranted the extensive industrial contacts of the project

consortium (see embedding), in particular in the area of conformance testing.

Test tool implementation. Starting point for tool development is the formal test tool TorX. TorX integrates automatic test generation and automatic test execution in an on-the-fly manner based on the theory of quiescent refusal testing; see above.

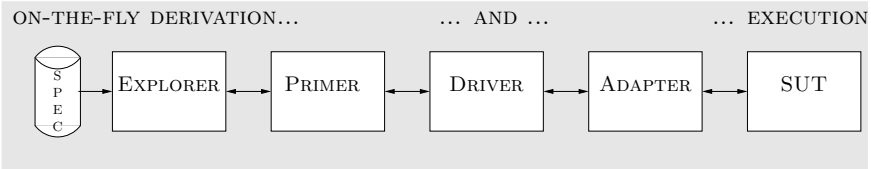

Figure 1: TORX tool architecture

The basic architecture of TorX consists of four components (Figure 1). A specification is fed into the Explorer; the explorer explores the state space of the specification. The Primer calculates *test primitives*: the events of the tests that will be executed. It implements the test generation algorithm of the quiescent refusal testing theory using the state exploration functions of the Explorer. The Driver keeps the on-the-fly test running and controls the other components. The Adapter is an important, but application specific part: it en- and decodes abstract primitives from and to application specific formats. The Sut is the *System under Test*. The left-hand side constitutes the formal part: the Explorer and Primer manipulate formal objects, such as the specification, to generate the tests. The right-hand side constitute the concrete part: the Sut is the real system that is being tested, which can be any piece of software, or hardware, or (usually) a combination of these. The Adapter transforms the abstract objects, e.g. an abstract data value, into a form which the Sut can understand, e.g a bit-string, and vice versa.

To extend TorX with realtime testing and data-oriented extension different modules are involved, both on the formal and on the concrete level.

For realtime test generation the Uppaal engine should be integrated into TorX, replacing or extending the current Explorer and Primer. This requires enriched interfaces between the Uppaal engine and the other components to convey realtime information. Also new techniques for test selection must be implemented.

For realtime test execution it is important to make TorX fast enough for on-the-fly selection of test data and for handling realtime responses. Many different techniques, such as from Real Time Operating Systems (RTOS), distributed implementations and pre-computations (look ahead), will be needed to make TorX fast enough, and to adapt the Adapter to deal with realtime stimuli and responses.

For data-oriented test generation TorX is extended with modules supporting test data selection. This includes computations on data, data selection strategies and, probably, constraint-solving or theorem-proving functionality. Also for this extension interfaces must be upgraded to cater for the richer information to be communicated. If it turns out that functional languages are indeed a good solution then the Clean-environment is a candidate for integration with TorX.

For all tool development activities it is important to maintain as much as possible the open, modular architecture with clear and generic interfaces, and to reuse and integrate existing components and tools wherever possible. This will lead to a flexible tool structure where future extensions or replacements of components, e.g., replacing one data formalism with another, are easy to be made.

(c) **Scientific Interest**

The scientific interest of the project is clear: a usable formal basis for realtime testing is long due. Independently, also a more rigorous treatment of test data selection is clearly a desirable goal. Although perhaps less fundamental, it is an essential ingredient to make the theory effective for testing real embedded software systems.

A lesson that can be learned from formal methods research in the past decade is that theory development and tool development must go hand in hand to obtain the best results, both in terms of theory and in terms of practical relevance. This makes the work on tool development an indispensable part of the project.

The proposed approach, based on the combination and adaptation of tool functionalities of the TorX and Uppaal tool sets will also create a very interesting link between realtime testing and realtime model checking, echoing the existing trend to view model checking and model-based testing as closely related activities. This will also create a natural opportunity to link other Uppaal-related developments, such as cost-optimal search techniques [LBB+01], with testing.

The proposed project is urgent in at least two ways. First, there exists a real need for formally supported testing of realtime embedded software systems, which should be addressed. Second, the proposed research is urgent in terms of the current availability of the necessary expertise. The Côte-de-Resyste project that has developed the TorX tools et will finish by mid-2002. In order to profit from the expertise that was built up during this project, STRESS should ideally start up in the second half of this year.

(d) **Related Work**

An important part of the academic literature on testing is about the (automatic) derivation of tests from (formal) specifications or models. The first approaches to testing the control-flow of communication protocols and embedded systems were inspired by hardware testing. These approaches use Finite State Machines (FSM – Mealy machines) to model the behaviour of the system. From an FSM tests are generated following algorithms known as 'checking sequences', 'W-method', 'UIO' and others [Cho78, BU91, LY96]. They mainly work for deterministic systems without data (although some extensions in these directions have been investigated). Their applicability to software testing is limited because of their restricted modelling power, mainly due to lack of compositionality. Moreover, the completeness claims of these methods with respect to testing critically depend on the (unrealistic) assumption that the number of states of any implementation never exceeds the number of states of the specification. Finally, we showed experimentally that these methods perform less than other testing methods [HFT00], and will therefore not consider FSM testing for this project.

Most of the current research on testing control-flow aspects concentrates around labelled transitions systems. It has its basis in the formal theory of testing equivalences by Hennessy and De Nicola and one of its main results are the theory of quiescent refusal testing and **ioco**-testing, as described in the previous sections; see also [BT01] for an overview of these developments.

With the increasing interest in model-based test derivation there are test generation tools, both academic and industrial, starting to emerge. One of the first ones was 'The RNL Conformance Kit' developed by the Dutch PTT and based on the FSM theory [BKKW90]. Later this tool was adopted by Philips and incorporated into Phact [FMMW98, HFT00].

Test tools based on the theory of quiescent refusal testing (**ioco**-testing) include Tveda [Pha94], Tgv [JM99], Testcomposer [KJG99], TestGen (Stirling) [HT99]. and TorX [BFV+99]. Related tools are Autolink (part of the Telelogic TAU tool set [SEK+98]), Agatha (using SDL and StateMate) [LRG01], SaMsTaG [GSDD97], STG [CJRZ01], and RT-Tester [PS97]. Except TorX, all these tools use the batch-approach as described in the previous sections. On-the-fly testing was also used for testing IBM's CICS system [Gri99].

In the area of test execution there are many, mostly commercial tools available, such as protocol testers and analysers, *capture & playback* tools, spreadsheet-based approaches (test are manually specified and written in a standard spreadsheet from which they are read and interpreted by test execution tools [BK99]), and code-coverage tools (which calculation implementation-

code coverage which is to be distinguished from specification coverage measures as used in the previous sections). For an overview of such test tools see [Sit00].

For testing based on timed automata, most other proposals are based on a discrete time interpretation or a discretized state space, combined with traditional FSM-based checking sequences for deterministic finite state machines [SVD97, ENDK98, CO00]. Besides the conceptual limitations as described above, this approach suffers severely from the resulting very large state spaces. Another approach is test purpose based derivation [CKL98, SS01] where a test case is constructed from an explicit scenario to be tested and a specification. None of these apply the efficient symbolic reachability analysis techniques of UPPAAL, and in addition have problems with soundness and completeness.

Data selection techniques for traditional sequential programs based on heuristics – equivalence partitioning, boundary value analysis – are already described in the first serious book on testing [Mye79]. These principles form the basis for some formal approaches to testing based on Z [Hie97, BCM00], usually by using some disjunctive normal form; cf. RTCAT.

A more systematic approach to testing data aspects is based on abstract data types [BGM91, Gau95]. A tool – LOFT – accompanies this approach [Mar95].

The tool STG is one of the first ones to combine control-flow testing and data testing in a non-trivial manner [Rus01, CJRZ01]. It restricts the data type language to something which is completely decidable, viz. Presburger arithmetic. Although this severely limits its applicability and also the use of some kind of trace preorder as the implementation relation is not so realistic, it is a first step towards symbolic test generation.

Other sources of inspiration for integration of data and process calculi are found in the work on value-passing processes [Lin93, IL01].

An approach to testing properties of functional programs, with a corresponding tool, is described in [CH00]. It is a testing solution to the kind of problems that Sparkle tries to conquer via theorem proving.

(e) **Local Embedding**

At the University of Twente the proposed research is part of the Formal Methods and Tools research theme of the Centre for Telematics and Information Technology. Formal methods based testing is one of the focal points of this theme. Part of the CTIT is the Twente Embedded Systems Initiative (TESI), where Formal Methods and Tools is one of the 6 core research groups.

The work will take place in an environment with a number of closely related research activities, such as the PROGRESS project ATOMYSTE (action refinement in embedded systems testing), the PROGRESS project HaaST (Verification of Hard and Softly Timed Systems), the NWO projects CASH (Compositional Analysis and Specification of Hybrid Systems, with the group of Van der Schaft at Twente), and SPACE (Specification-based Performability Checking). International projects are the EU projects AMETIST (Advanced Methods for Timed Systems) and ARTIST (Advanced Realtime Systems), and the Dutch-German VOSS (Validation of Stochastic Systems). There also exists an intensive research collaboration with the group of Larsen at Aalborg (Denmark).

At Nijmegen University the work will be embedded in the research programme of the Software Technology group. This group recently started a new line of research on Model Based Software Testing. Of special interest for the data-oriented testing part of the project are the group's research activities on functional languages in general, and the Clean System in particular. This includes the dedicated proof system Sparkle which is currently under development. The Software Technology group is involved in ATOMYSTE and will participate as subcontractor in ARTIST mentioned above. The Software Technology group has strong links with the Technical Applications group of Vaandrager via the common research theme "Quality of Software". The Technical Applications group has substantial expertise in the analysis of realtime systems including testing and the use of UPPAAL. Also experience with theorem provers is available. The group is a partner in the in the HaaST project, the EU project AMETIST and VOSS mentioned above.

Through the various projects there is contact and collaboration with various industries in the area of software testing, such as CMG Government B.V., CMG Transport, Trade & Industry, Ericsson Aachen, InterPay B.V., Lucent Technologies, Philips and Siemens Aalborg.

7. **Work Programme**

The work will be carried out in the form of two PhD research projects, one on realtime extensions at Twente, and one on testing with complex data at Nijmegen. The implementation work will be supported by a programmer, who will be stationed at Twente, where until now all TorX development work has taken place.

We anticipate a close form of interaction between the persons and groups. This will be needed to ensure the compatibility of the approaches to realtime and data and their smooth integration in the TorX extension. On top of this, substantial collaboration between the sites on the common issues, in particular that of test selection for symbolic data representations, is anticipated.

Given the already strong collaboration of the Formal Methods and Tools group at Twente with Nijmegen (AMETIST, ARTIST, ATOMYSTE, HaaST, VOSS) the organization of this joint project will be fairly straightforward. In practical terms it will be implemented in the form of regular joint project meetings (every 2 months) and (more) frequent mutual site visits by the PhD students and the programmer.

The outline of the activities of the PhD students, in addition to their IPA educational programme, is as follows:

AIO 1: realtime extensions

- **year 1:**
  - study relevant background material
    (formal testing theory, timed automata, TorX, Uppaal, RTCAT);
  - simple extensions of quiescent refusal testing
- **year 2:**
  - quiescent refusal testing for interesting classes of input-enabled timed automata;
  - initial development of test derivation/selection algorithms (realtime);
  - study of (realtime aspects of) test-execution environments.
- **year 3:**
  - efficient test derivation/selection algorithms (realtime);
  - test execution strategies (realtime);
  - application case study
- **year 4:** wrap-up, writing of thesis

AIO 2: testing with complex data

- **year 1:**
  - study relevant background material
    (formal testing theory, test data selection methods, symbolic transition systems, TorX, Clean);
  - study and select data specification techniques.
- **year 2:**
  - integration of structured data representations in formal testing framework;
  - initial development of test derivation/selection algorithms (data);
  - study of (data aspects of) test-execution environments.
- **year 3:**
  - efficient test derivation/selection algorithms (data);

  - test execution strategies (data);
  - application case study
- **year 4:** wrap-up, writing of thesis

A programmer will be needed to carry out most of the implementation work and to coordinate the practical aspects of the work on extending TorX. The main tool functionalities will be implemented in two rounds, the second versions being based on the evaluation of the application case studies carried out in third year.

Based on previous experience with formal methods related tool development, we will be looking for a programmer with an academic background who can handle the advanced and rather abstract concepts that are central to the project.

The outline of the programmer's work is:

Programmer

- **year 1:**
  - familiarization with TorX and Uppaal;
  - familiarization with RTOS;
  - time-optimization of TorX component implementations
- **year 2:**
  - revision of TorX tool architecture
  - interface adaptations for realtime & data testing
- **year 3:**
  - implementation of symbolic test derivation/selection;
  - realtime test execution support;
  - tool support application case studies
- **year 4:**
  - implementation of symbolic test derivation/selection (version 2);
  - realtime test execution support (version 2);
  - tool optimization, documentation, maintenance

8. **Expected use of Instrumentation.**

   3 workstations (1 x programmer, 2 x AIO)

9. **Literature.**

   Relevant publications of the Twente team are: [Bri88, Bri93, BT01, LPY97, LBB+01]
   Relevant publications of the Nijmegen team are: [PE93, Tre96b, Tre99, BT01, MEP02].

10. **Requested Budget**

AIO/postdoc	2 AIO bench fee	2 x 129.879 2 x 4.538	259.758 9.076
programmer	1 ac. 4 yrs	1 x 222.731	222.731
comp. equipment	3 workstations	3 x 5.000	15.000
total			506.565

References

[AFH94] R. Alur, L. Fix, and T.A. Henzinger. Event-Clock Automata: A Determinizable Class of Timed Automata. In D.L. Dill, editor, *Sixth Int. Conference on Computer Aided Verification – CAV'94*, volume 818 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[BCM00] S. Burton, J. Clark, and J. McDermid. Testing, Proof and Automation. An Integrated Approach. In *Procs. of the* 1^st *Int. Workshop of Automated Program Analysis, Testing and Verification*. ACMSIGSOFT IEEE Computer Science, June 2000.

[Bei90] B. Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, 1990. Second Edition.

[BFHV01] A. Belinfante, J. Feenstra, L. Heerink, and R.G. de Vries. Specification Based Formal Testing: The EasyLink Case Study. In *Progress 2001 –* 2^{nd} *Workshop on Embedded Systems*, pages 73–82, Utrecht, The Netherlands, October 18 2001. STW Technology Foundation. Full version: Internal Research Report, Philips Research Laboratories Eindhoven, The Netherlands, 2001.

[BFV+99] A. Belinfante, J. Feenstra, R.G. de Vries, J. Tretmans, N. Goga, L. Feijs, S. Mauw, and L. Heerink. Formal Test Automation: A Simple Experiment. In G. Csopaki, S. Dibuz, and K. Tarnay, editors, *Int. Workshop on Testing of Communicating Systems 12*, pages 179–196. Kluwer Academic Publishers, 1999.

[BGM91] G. Bernot, M. G. Gaudel, and B. Marre. Software testing based on formal specifications: a theory and a tool. *Software Engineering Journal*, 1991(November):387–405, 1991. Also: Rapport de Recherche 581, L.R.I., Université de Paris-Sud.

[BK99] H. Buwalda and M. Kasdorp. Getting Automated Testing under Control. *Software Testing & Quality Engineering*, 1(6):38–44, November/December 1999.

[BKKW90] S.P. van de Burgt, J. Kroon, E. Kwast, and H.J. Wilts. The RNL Conformance Kit – Tools for Automatic Test Suite Generation. In J. de Meer, W. Effelsberg, and L. Mackert, editors, *Second Int. Workshop on Protocol Test Systems*, pages 279–294. North-Holland, 1990.

[Bri88] E. Brinksma. A theory for the derivation of tests. In S. Aggarwal and K. Sabnani, editors, *Protocol Specification, Testing, and Verification VIII*, pages 63–74. North-Holland, 1988. Also: Memorandum INF-88-19, University of Twente, The Netherlands.

[Bri93] E. Brinksma. On the coverage of partial validations. In M. Nivat, C.M.I. Rattray, T. Rus, and G. Scollo, editors, *AMAST'93*, pages 247–254. BCS-FACS Workshops in Computing Series, Springer-Verlag, 1993.

[BT01] E. Brinksma and J. Tretmans. Testing Transition Systems: An Annotated Bibliography. In F. Cassez, C. Jard, B. Rozoy, and M.D. Ryan, editors, *Modeling and Verification of Parallel Processes –* 4^{th} *Summer School MOVEP 2000*, volume 2067 of *Lecture Notes in Computer Science*, pages 187–195. Springer-Verlag, 2001.

[BU91] B. S. Bosik and M. Ü. Uyar. Finite state machine based formal methods in protocol conformance testing: From theory to implementation. *Computer Networks and ISDN Systems*, 22(1):7–33, 1991.

[CG97] O. Charles and R. Groz. Basing Test Coverage on a Formalization of Test Hypotheses. In M. Kim, S. Kang, and K. Hong, editors, *Int. Workshop on Testing of Communicating Systems 10*, pages 109–124. Chapman & Hall, 1997.

[CH00] K. Claessen and J. Hughes. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. In *International Conference on Functional Programming 2000*. ACM Press, 2000. `http://www.cs.chalmers.se/ rjmh/QuickCheck/`.

[Cho78] T.S. Chow. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, 4(3):178–187, 1978.

[Chr01] P. Christian. Specification Based Testing with IDL and Formal Methods: A Case Study in Test Automation. Master's thesis, University of Twente, Enschede, The Netherlands, 2001.

[CJRZ01] D. Clarke, T. Jéron, V. Rusu, and E. Zinovieva. STG: A Tool for Generating Symbolic Test Programs and Oracles from Operational Specifications. In *Joint 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9)*, 2001.

[CKL98] R. Castanet, Ousmane Koné, and Patrice Laurençot. On the fly test generation for real-time protocols. In *International Conference in Computer Communications and Networks*, Lafayette, Lousiana, USA, October 12-15 1998. IEEE Computer Society Press.

[CO00] Rachel Cardell-Oliver. Conformance Testing of Real-Time Systems with Timed Automata. *Formal Aspects of Computing*, 12(5):350–371, 2000.

[COR+95] J. Crow, S. Owre, J. Rushby, N. Shankar, and M. Srivas. A Tutorial Introduction to PVS. Tutorial at WIFT'95: Workshop on Industrial-Strength Formal Specification Techniques, April 1995. Boca Raton, Florida, USA.

[DN87] R. De Nicola. Extensional equivalences for transition systems. *Acta Informatica*, 24:211–237, 1987.

[DNH84] R. De Nicola and M.C.B. Hennessy. Testing Equivalences for Processes. *Theoretical Computer Science*, 34:83–133, 1984.

[ENDK98] Abdeslam En-Nouaary, Rachida Dssouli, and Ferhat Khendek. Timed Test Cases Generation Based on State Characterization Technique. In *19th IEEE Real-Time Systems Symposium (RTSS'98)*, pages 220–229, December 2–4 1998.

[FGST02] L.M.G. Feijs, N. Goga, Mauw S., and J. Tretmans. Test Selection, Trace Distance and Heuristics. In H. König, I. Schieferdecker, and A. Wolisz, editors, *TestCom 2002*. Kluwer Academic Publishers, 2002. To appear.

[FMMW98] L.M.G. Feijs, F.A.C. Meijs, J.R. Moonen, and J.J. Wamel. Conformance testing of a multimedia system using PHACT. In A. Petrenko and N. Yevtushenko, editors, *Int. Workshop on Testing of Communicating Systems 11*, pages 193–210. Kluwer Academic Publishers, 1998.

[Gau95] M.-C. Gaudel. Testing can be formal, too. In P.D. Mosses, M. Nielsen, and M.I. Schwartzbach, editors, *TAPSOFT'95: Theory and Practice of Software Development*, pages 82–96. Lecture Notes in Computer Science 915, Springer-Verlag, 1995.

[GQ99] I.S. Graham and L. Quin. *XML Specification Guide*. Wiley Computer Publishers, 1999.

[Gri99] I. Griffiths. Executing Dynamic Self Determining Test Cases. In *EuroSTAR'99:* 7^{th} *European Int. Conference on Software Testing, Analysis & Review*, Barcelona, Spain, November 8–12, 1999. EuroStar Conferences, Galway, Ireland.

[GSDD97] J. Grabowski, R. Scheurer, Z.R. Dai, and Hogrefe. D. Applying SaMsTaG to the B-ISDN protocol SSCOP. In M. Kim, S. Kang, and K. Hong, editors, *Int. Workshop on Testing of Communicating Systems 10*, pages 397–415. Chapman & Hall, 1997.

[Hee98] L. Heerink. *Ins and Outs in Refusal Testing*. PhD thesis, University of Twente, Enschede, The Netherlands, 1998.

[HFT00] L. Heerink, J. Feenstra, and J. Tretmans. Formal Test Automation: The Conference Protocol with PHACT. In H. Ural, R.L. Probert, and G. von Bochmann, editors, *Testing of Communicating Systems – Procs. of TestCom 2000*, pages 211–220. Kluwer Academic Publishers, 2000.

[Hie97] R.M. Hierons. Testing from a Z Specification. *Software Testing, Verification and Reliability*, 7:19–33, 1997.

[Hol91] G.J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall Inc., 1991.

[HT99] J. He and K.J. Turner. Protocol-Inspired Hardware Testing. In G. Csopaki, S. Dibuz, and K. Tarnay, editors, *Int. Workshop on Testing of Communicating Systems 12*, pages 131–147. Kluwer Academic Publishers, 1999.

[IL01] A. Ingólfsdóttir and H. Lin. A Symbolic Approach to Value-Passing Processes. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 7, pages 427–478. Elsevier Science B.V., 2001.

[ISO89] ISO. *Information Processing Systems, Open Systems Interconnection, LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*. International Standard IS-8807. ISO, Geneve, 1989.

[ISO90] ISO. *Information Processing Systems, Open Systems Interconnection, Specification of Abstract Syntax Notation One (ASN.1)*. International Standard IS-8824. ISO, Geneve, 1990.

[ISO91] ISO. *Information Technology, Open Systems Interconnection, Conformance Testing Methodology and Framework*. International Standard IS-9646. ISO, Geneve, 1991. Also: CCITT X.290–X.294.

[JM99] T. Jéron and P. Morel. Test generation derived from model-checking. In N. Halbwachs and D. Peled, editors, *Computer Aided Verification CAV'99*, pages 108–121. Lecture Notes in Computer Science 1633, Springer-Verlag, 1999.

[KJG99] A. Kerbrat, T. Jéron, and R. Groz. Automated Test Generation from SDL Specifications. In R. Dssouli, G. von Bochmann, and Y. Lahav, editors, *SDL'99, The Next Millennium – Proceedings of the* 9^{th} *SDL Forum*, pages 135–152. Elsevier Science, 1999.

[Lan90] R. Langerak. A testing theory for LOTOS using deadlock detection. In E. Brinksma, G. Scollo, and C. A. Vissers, editors, *Protocol Specification, Testing, and Verification IX*, pages 87–98. North-Holland, 1990.

[LBB+01] K.G. Larsen, G. Behrmann, E. Brinksma, A. Fehnker, P. Petterson, and J. Romijn. As Cheap as Possible: Efficient Cost-Optimal Reachability for Priced Timed Automata. In G. Berry, H. Comon, and A. Finkel, editors, *Computer Aided Verification – CAV 2001*, volume 2102 of *Lecture Notes in Computer Science*, pages 493–505. Springer-Verlag, 2001.

[Lin93] H. Lin. A verification tool for value-passing processes. In A. Danthine, G. Leduc, and P. Wolper, editors, *Protocol Specification, Testing, and Verification XIII*. North-Holland, 1993.

[LPY97] K.G. Larsen, P. Petterson, and W. Yi. UPPAAL in a Nutshell. *Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

[LRG01] D. Lugato, N. Rapin, and J.-P. Gallois. Verification and Test Generation for SDL industrial Specifications with the AGATHA toolset. In P. Pettersson and S. Yovine, editors, *Workshop on Real-Time Tools 2001 – An Affiliated workshop to CONCUR'2001*, 2001.

[LT89] N.A. Lynch and M.R. Tuttle. An introduction to Input/Output Automata. *CWI Quarterly*, 2(3):219–246, 1989. Also: Technical Report MIT/LCS/TM-373 (TM-351 revised), Massachusetts Institute of Technology, Cambridge, U.S.A., 1988.

[LY96] D. Lee and M. Yannakakis. Principles and methods for testing finite state machines – a survey. *The Proceedings of the IEEE*, 84(8):1090–1123., August 1996.

[Mar95] B. Marre. LOFT: A tool for assisting selection of test data sets from algebraic specifications. In P.D. Mosses, M. Nielsen, and M.I. Schwartzbach, editors, *TAPSOFT'95: Theory and Practice of Software Development*, pages 799–800. Lecture Notes in Computer Science 915, Springer-Verlag, 1995.

[MEP02] M. de Mol, M. van Eekelen, and R. Plasmeijer. Theorem Proving for Functional Programmers. In Th. Arts and M. Mohnen, editors, *13th International Workshop on the Implementation of Functional Languages – IFL'01*, Lecture Notes in Computer Science. Springer-Verlag, 2002.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[MV95] M. Mori and S.T. Vuong. On finite covering of infinite spaces for protocol test selection. In S.T. Vuong and S.T. Chanson, editors, *Protocol Specification, Testing, and Verification XIV*, pages 237–251. Chapman & Hall, 1995.

[Mye79] G.J. Myers. *The Art of Software Testing*. John Wiley & Sons Inc., 1979.

[Nes90] J.R. Nestor. *IDL: The Language and its Implementation*. Prentice Hall, 1990.

[NH84] R. De Nicola and M.C.B Hennessy. Testing Equivalences for Processes. *Theoretical Computer Science*, 34:83–133, 1984.

[Nie00] Brian Nielsen. *Specification and Test of Real-Time Systems*. PhD thesis, Department of Computer Science, Aalborg University, Denmark, april 2000.

[NS00] Brian Nielsen and Arne Skou. Automated Test Generation Timed Automata. In *21st IEEE Real-Time Systems Symposium 2000 Work in Progress-Session*, Walt Disney World, Orlando, Florida, USA, November 2000. IEEE.

[NS01a] Brian Nielsen and Arne Skou. Automated Test Generation Timed Automata. In Tiziana Margaria and Wang Yi, editors, *TACAS 2001 - Tools and Algorithms for the Construction and Analysis of Systems*, pages 343–357, Genova, Italy, April 2001.

[NS01b] Brian Nielsen and Arne Skou. Test Generation for Time Critical Systems: Tool and Case Study. In *13th Euromicro Conference on Real-Time Systems*, pages 155–162, Delft, The Netherlands, June 2001.

[NS02] Brian Nielsen and Arne Skou. Automated Test Generation from Timed Automata. *International Journal on Software Tools for Technology Transfer (STTT)*, 2002. Invited Contribution. Under review.

[PE93] M.J. Plasmeijer and M.C.J.D. van Eekelen. *Functional Programming and Parallel Graph Rewriting*. Addison Wesley, 1993.

[PE99] R. Plasmeijer and M. van Eekelen. Keep it Clean: A Unique Approach to Functional Programming. *ACM Sigplan Notices*, June 1999.

[PE01] R. Plasmeijer and M. van Eekelen. Clean Language Report, Version 2.0. url: `www.cs.kun.nl/ clean`, University of Nijmegen, Nijmegen, The Netherlands, 2001.

[Pha94] M. Phalippou. *Relations d'Implantation et Hypothèses de Test sur des Automates à Entrées et Sorties*. PhD thesis, L'Université de Bordeaux I, France, 1994.

[Phi87] I. Phillips. Refusal testing. *Theoretical Computer Science*, 50(2):241–284, 1987.

[PS97] J. Peleska and M. Siegel. Test automation of safety-critical reactive systems. *South African Computer Jounal*, 19:53–77, 1997.

[Rus01] V. Rusu. Verifying Automata with Presburger Arithmetic and Uninterpreted Function Symbols. In *Int. Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001)*, Lecture Notes in Computer Science. Springer-Verlag, 2001. Submitted.

[SEK+98] M. Schmitt, A. Ek, B. Koch, J. Grabowski, and D. Hogrefe. – AUTOLINK – Putting SDL-based Test Generation into Practice. In A. Petrenko and N. Yevtushenko, editors, *Int. Workshop on Testing of Communicating Systems 11*, pages 227–243. Kluwer Academic Publishers, 1998.

[Sit00] M. Siteur. *Testen met Testtools – Slapend Werken*. Academic Service, 2000. url: `http://www.siteur.myweb.nl/`. (In Dutch).

[Sof97] Software Tools for Technology Transfer. Special Section on Timed and Hybrid Systems, December 1997.

[SS01] Hacène Fouchal Sébastian Salva, Eric Petitjean. A simple Approach to Testing Timed Systems. In Ed Brinksma and Jan Tretmans, editors, *Formal Approaches to Testing of Software, FATES'01*, Aalborg, Denmark, august 2001.

[STW96] Dutch Technology Foundation STW. *Côte de Resyste* – COnformance TEsting of REactive SYSTEms. Project proposal STW TIF.4111, University of Twente, Eindhoven University of Technology, Philips Research Laboratories, KPN Research, Utrecht, The Netherlands, 1996. `http://fmt.cs.utwente.nl/CdR`.

[SVD97] J. Springintveld, F. Vaandrager, and P.R. D'Argenio. Testing Timed Automata. TR CTIT 97-17, University of Twente, 1997. To appear in *Theoretical Computer Science*.

[Tre96a] J. Tretmans. Conformance testing with labelled transition systems: Implementation relations and test generation. *Computer Networks and ISDN Systems*, 29:49–79, 1996.

[Tre96b] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software—Concepts and Tools*, 17(3):103–120, 1996. Also: Technical Report No. 96-26, Centre for Telematics and Information Technology, University of Twente, The Netherlands.

[Tre99] J. Tretmans. Testing Concurrent Systems: A Formal Approach. In J.C.M. Baeten and S. Mauw, editors, *CONCUR'99 – 10^{th} Int. Conference on Concurrency Theory*, volume 1664 of *Lecture Notes in Computer Science*, pages 46–65. Springer-Verlag, 1999.

[Vaa91] F. Vaandrager. On the relationship between process algebra and Input/Output Automata. In *Logic in Computer Science*, pages 387–398. Sixth Annual IEEE Symposium, IEEE Computer Society Press, 1991.

[VBF02] R.G. de Vries, A. Belinfante, and J. Feenstra. Automated Testing in Practice: The Highway Tolling System. In H. König, I. Schieferdecker, and A. Wolisz, editors, *TestCom 2002*. Kluwer Academic Publishers, 2002. To appear.

[YJ94] W. Yi and B. Jonsson. Decidability of Timed Language-Inclusion for Networks of Real-Time Communicating Sequential Processes. In 14^th *Conference on Foundations of Software Technology and Theoretical Computer Science*, Madras, India, 1994.