Testability Analysis for Graphically Described Algorithms of Reactive Systems

H V Do, C Robach, M Delaunay, J.-S Cruz

Testability Analysis for Graphically Described Algorithms of Reactive Systems

H. V. Do[1], C. Robach[1], M. Delaunay[2], J.-S. Cruz[3]

1: LCIS – ESISAR, BP 54, 50 rue Barthélémy de Laffemas, 26902 Valence Cedex 09, France
2: LSR – IMAG, BP72, 681 rue de la Passerelle, 38402 St Martin d'Hères, France
3: MBDA, BP 84, 2 rue Béranger, 92323 Châtillon Cedex, France

Abstract: Reactive Real-Time Systems require very high level of confidence. The validation, which ensures the confidence of these systems, is often difficult and expensive. A testability analysis at the design phase of a system can identify parts that are difficult for system testing. Such an analysis helps the designer to improve the design, reduces the cost of the validation, and increases the confidence of the system.

During the development of Reactive Real-Time systems, graphic environments are often used to design systems. Our approach allows analyzing automatically the testability of systems from their graphical descriptions.

Keywords: Reactive Real-Time system, Data-Flow design, Testability Analysis

1. Introduction

Reactive Real-Time Systems are largely utilized in many safety-critical domains, for instance: avionics, automotive, aerospace. For the development of these sorts of systems, validation plays an important role to ensure confidence in these systems. The validation process is divided into two main activities: the proof of some parts of the system, which verifies safety properties of these parts of system; and the testing phase that reveals faults in the system. Testing of complex reactive real-time systems is very expensive. Hence, a testability analysis, which takes into account the validation as soon as the specification phase, is very promising.

Many languages and graphical environments are used in the development of reactive real-time systems: the Simulink [1] and Scicos [2] languages are utilized classically to describe discrete (or/and continuous) data's operations, the StateChart [3] language useful for constructing behavior models. UML 2.0 notations [4] define the numerical system and can be used to describe the software, or software and hardware components. The AADL [5] language makes the mapping on the hardware: this language using a description of the software, or of the software and hardware and to include the interface between those components with the tasks, the processes, the buses, and the communications. In this way, synchronous languages are successfully used: the Lustre [6] language for data-flow design, the Signal [7] language for the control of data-flow (consistency with a GALS [7] approach) for system level design.

In order to simplify the development process, graphical environments (such as Scade/Lustre, Scade/Esterel, Sildex/Signal or Polychrony/Signal, Simulink/Matlab, StateFlow/StateChart or Statemate/Statechart) supply important utilities: simulators, provers, certified code generators, test data generators. In this sort of environment, recently the Advanced System Development Environment (ASDE) [8] provides a complete set of CASE tools, which support development activities and meet the high dependability needs of safety-critical reactive real-time systems that support DO178B requirement. Resuming, graphical environments facilitate the complexity control of system by providing a graphical hierarchical view. These environments allow designers: to describe complex algorithms and their data-flow, control-flow; to simulate and/or prove them. In this way, code can be generated from validated algorithms by certified code generators. Hence, the overall design process is reduced.

In the algorithms design, designers can use these graphical environments to describe computation components and the data-flow or control-flow of systems. A data-flow or control-flow design is a diagram of operators or subsystems. A subsystem is also a data-flow design. However, in real complex systems, it is difficult to analyse the weaknesses of their architecture, mainly because faults can be hidden inside subsystems. In order to solve this difficulty, we propose an automated method of analysing the testability of data-flow designs of such reactive systems, which are designed by using graphical environments (Scade, Sildex, Simulink, Scicos, Rapsody or Rose / UML 2.0, Stood/AADL). Our method, which is based on the SATAN technology (System's Automatic Testability Analysis) [9], consists in analysing the data flow of systems. This technology allows: 1) identifying elementary functions of the system; 2) determining test objectives through strategies; 3) computing testability measures of each component in the system. The testability measures in the SATAN technology are based on the information theory [10].

In order to reach our objectives, we use an Information Transfer Graph (ITG) to model a data-flow design. From this ITG, once elementary functions of the system are identified, test objectives can then be obtained by applying one of two strategies: the Start-Small strategy and the Multiple-Clue strategy.

The testability is defined as a combination of two measures: controllability and observability. We use an Information Transfer Net (ITN) to simulate the information circulation in the data-flow design. Thus, the testability measures are calculated by evaluating the information loss in the ITN, where each operator contributes to this loss. The information loss of an operator can be calculated exactly (for logical operators) or statistically (for other operators).

This paper proceeds as follows: section 2 introduces some criteria of reactive real-time systems, which affect the testability analysis. Section 3 presents the SATAN technology. Section 4 outlines our statistical evaluation of ILC. Section 5 details our testability analysis process of data-flow designs. A case study is illustrated in Section 6. Finally, some conclusions and perspectives are given in Section 7.

2. Criteria of Reactive Real-Time Systems

In this section, we study some criteria of reactive real-time systems that have influence on the use of the SATAN technology.

A reactive system can be viewed as a set of functions that specify the relationship between its inputs, outputs, states and time (see Figure 1).

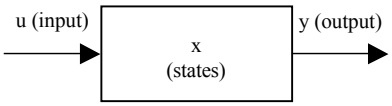

Figure 1: Reactive real-time system

The system functions include: an output function f_o, an update function f_u. The f_o function computes the system's output from time, states and inputs. The f_u function computes the future values of the system's states from the current time, inputs and state.

The states of a reactive system are related to the automat modelling that system. In our method, we analyze just the data flow of systems. So we concentrate on the output function f_o.

The synchronous approach [11] is largely used in safety-critical reactive system. In this approach, the time is divided into discrete instants defined by a global clock. At instant t, the system receives input i_t from its external environment, and computes output o_t.

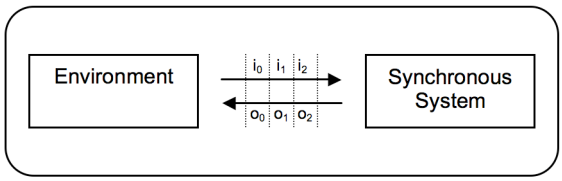

Figure 2: Operation of a synchronous system

The synchronous hypothesis expresses that the computation of the output values is made instantaneously at the same instant t. The operation of a synchronous system is illustrated in Figure 2. Our approach analyzes the static aspect of synchronous systems, i.e. the data-flow of a system at an instant t.

We use a graphically described algorithm called GRS as the main example in this paper. The GRS example has been designed in the Simulink environment. It contains eight inputs, a subsystem followed by several basic operators, and two outputs (see Figure 3).

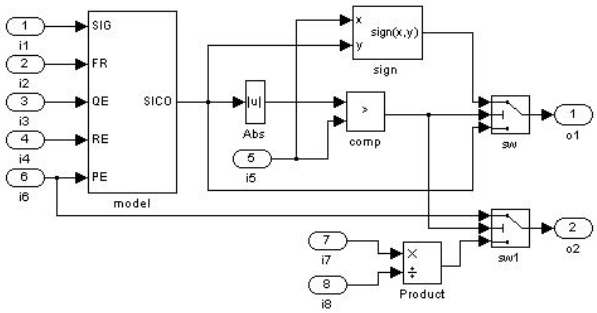

Figure 3: The GRS example

The algorithms described graphically have a hierarchical structure. Thus, a reactive system is generally modeled in different levels. In each level of specification, the system is a diagram of operators or subsystems. For example, the GRS design has two levels of specification. Because the SATAN technology supports one-level structure, before applying the SATAN technology, the system must be flattened into a one-level structure, as depicted in Figure 4.

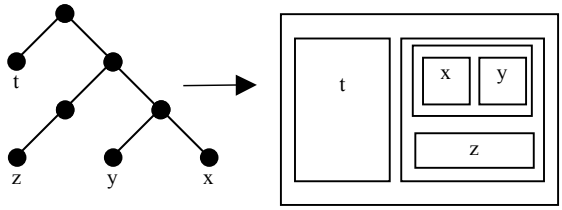

Figure 4: Flattening the hierarchical structure

3. The SATAN technology

The SATAN technology was originally proposed for hardware systems. It was then shown to be applicable for data-flow designs in [12, 13].

In the operator diagram of a system, each operator is associated with a testability model that represents the information flow of this operator. The diagram of operators is then transformed into a global model by combining each model of operators. The testability model is called *Information Transfer Graph* (or ITG).

The test objectives of a system can be determined from the information flows in the ITG.

3.1 Information Transfer Graph

An ITG is a bipartite directed graph defined by a set of places, transitions and edges. The places are:

- The inputs of the system, but also the control points of the system where testing data can be sent;
- The modules, which are functional modules of operators of the system;
- The outputs of the system, but also the points of observation of testing results.

The transitions between places express the conditions allowing the information transfer between those places, together with the transfer modes. There are three modes of information transfer (Figure 5):

- The junction mode: the destination place needs information from all source places.
- The attribution mode: the destination place needs information from one of several source places.
- The selection mode: the same information is sent from the source place to some destination places.

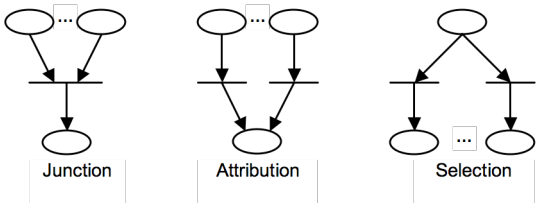

Figure 5: Information transfer modes

The edges connecting places and transitions represent the information media throughout the system. In the graphic representation, inputs and outputs are depicted by semicircles, modules by circles, and transitions by bars.

The ITG contains all control points, like a conventional control flow graph. And it also contains all relations between definitions and uses of variables, like a conventional data-flow graph. Hence, the ITG is suitable for data-flow designs.

The ITG associated to the GRS example is given in Figure 6.

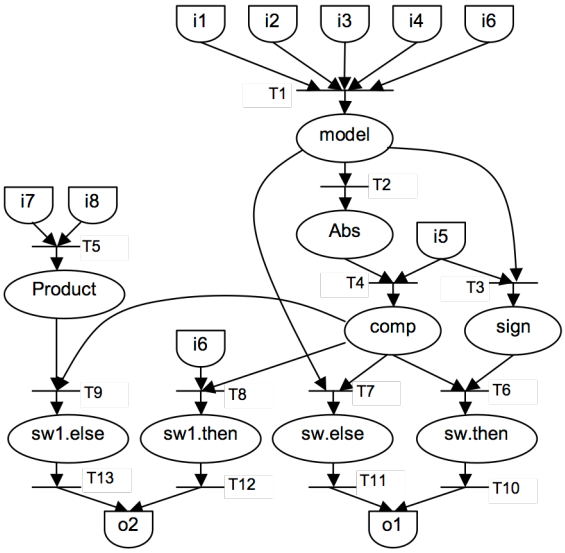

Figure 6: The ITG of the GRS example

3.2 Information Flows

An information flow (or *flow*) guides information from one or several inputs, through several modules and transitions, to one output. Hence, each flow corresponds to an elementary function of the system. Every flow is determined such that:

- If an output place of a junction is in the flow, all its predecessors belong to the flow;
- If an output place of an attribution is in the flow, one of its predecessors belongs to the flow;
- If an input place of a selection is in the flow, one of its successors belongs to the flow;
- No flow is the union of other flows.

Two important facts to consider:

1) All edges arriving at a place allow controlling all data necessary for testing this place;
2) All edges leaving from a place allow observing all results produced by this place.

All flows in an ITG are identified automatically by the SATAN technology.

The ITG of the top level of the GRS example contains four flows. Each flow F_i is described by all its modules and its output:

F1 = {*model, Abs, comp, sign, sw.then* | *o1*}

F2 = {*model, Abs, comp, sw.else* | *o1*}

F3 = {*model, Abs, comp, sw1.then* | *o2*}

F4 = {*model, Abs, comp, Product, sw1.else* | *o2*}

3.3 Test strategies

Once all flows of the ITG are identified, a test strategy can be applied to determine a set of test

objectives for the software. In fact, a test strategy is a way to choose flows, which respects the following criterion: every place in the graph must be activated at least once, to ensure the coverage of all modules.

Two test strategies are supported by the SATAN technology: the progressive structural strategy (*Start-Small*) and the cross-checking strategy (*Multiple-Clue*) [14].

The *Start-Small* strategy covers gradually the modules by choosing flows with an increasing number of covered modules. The main idea of this strategy is to minimize the effort of diagnosis. The first flow to be tested must contains the minimal quantity of modules. The next flow to be tested contains a minimal quantity of modules that are not activated yet. In this strategy, a new flow is tested only if all faults detected in previous flows are corrected, as depicted in Figure 7. This strategy is suitable for the progressive detection of faults during the validation process.

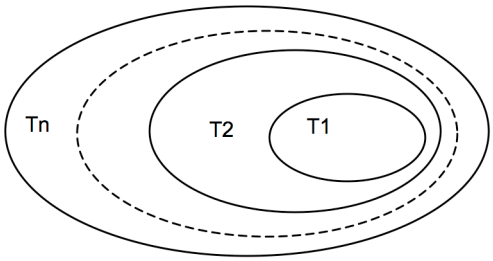

Figure 7: The Start-Small strategy

The *Multiple-Clue* strategy is based on choosing the minimum subset of flows that covers all modules. This strategy refers to the notion of differential matrix. In this strategy, all flows chosen are executed, and diagnostic is realized on possible fault information, see Figure 8. This strategy is suitable for diagnosis during maintenance.

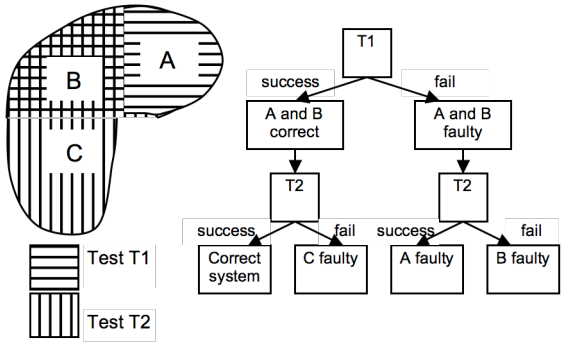

Figure 8: Analysis of testing results with Multiple Clue

These two strategies are complementary: the Start-Small strategy can be used to detect multiple faults; on the other hand, the Multiple-Clue strategy gives the best fault localisation for single fault (it can treat undistinguishable fault information found by the Start-Small strategy).

These two strategies allow the number of test objectives to be minimized while ensuring that all components of the system are activated. By applying these two strategies, the validation and the maintenance are taken into account very soon at the design phase.

3.4 Information Transfer Net

The Information Transfer Net (ITN) is used to simulate the transfer of information in the diagram of operators. The ITN is a weighted ITG, whose elements are associated with their information capacities.

The information capacities of inputs and outputs of the system are evaluated from their data types.

The capacity of a functional module is the product of the capacity of its output and the Information Loss Coefficient.

The capacity of an edge leaving from a place is equal to the capacity of that place.

The capacity of an edge arriving at an output place is equal to the capacity of this output place.

In the junction mode, the capacity of the edge that leaves the transition is the sum of capacities of all edges arriving at the transition.

In the selection mode, the capacity of each edge leaving from the source place is the capacity of the source place.

In the attribution mode, the capacity of the destination place is the maximum value of capacities of every edges arriving at that place.

3.5 Constructing ITG and ITN

The ITG of a design is built by concatenating elementary ITGs of its operators. An elementary ITG represents the data-flow of the corresponding operator.

The ITN of a design is built from the ITG and elementary ITNs of its operators. An elementary ITN represents the information circulation and information loss of the corresponding operator.

For example, the elementary ITG and ITN of the AND operator is presented in figure 9.

We use SATAN libraries to manage elementary ITGs and elementary ITGs of operators. These elements are obtained from the description and the use of operators. A SATAN library corresponds to a graphical library of the development environment.

3.6 Testability Measures

In this approach, the testability is based on the controllability and the observability of a module for

each flow that contains this module. The controllability expresses how ease the input values of an internal component can be controlled through the input values of the system. The observability expresses how ease the results of an internal component can be observed at the outputs of the system (Figure 10).

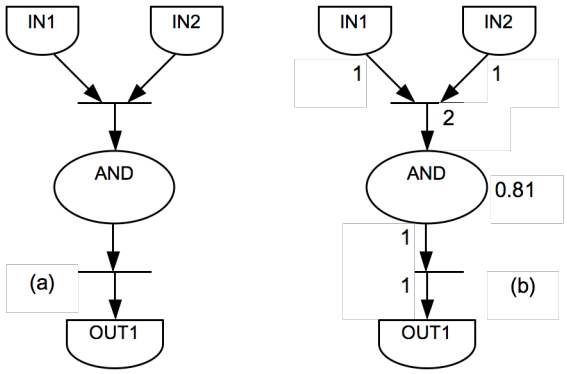

Figure 9: Elementary ITG and ITN of the AND operator.

With our approach, the SATAN technology computes testability measures only for flows chosen by a strategy, because the coverage of elementary functions is ensured.

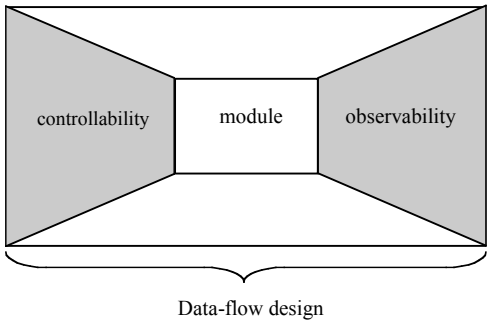

Figure 10: Controllability and observability of a module

Next, we describe the computation of testability measures. Suppose that F is a flow in the ITG, and M is a module in the flow F, the variables of F are denoted as follows:

- X_M is the input variable of module M
- Y_M is the output variable of module M
- I_F is the source variable of flow F
- O_F is the destination variable of flow F

We compute the controllability measure of the module M in the flow F as follow:

If module M is isolated, all the possible combinations of its inputs can be produced. And the information capacity of M is the maximum information of its inputs: C (X_M).

If the module M belongs to the flow F, the information quantity that can be brought to its inputs is: $T(O_F ; X_M)$. This information quantity is the maximal flow from the inputs O_F to the module M. The controllability of the module M in F is computed by the following equation:

$$Cont_F(M) = \frac{T(O_F;X_M)}{C(X_M)}$$

In the same way, we compute the observability measures of the module M in the flow F as follow: the total information quantity that the module M can produce is $C(Y_M)$. The maximum information quantity that the module M can deliver to the destinations of F is: $T(Y_M ; O_F)$. The observability of the module M in F is computed by the following equation:

$$Obs_F(M) = \frac{T(Y_M;O_F)}{C(Y_M)}$$

4. Statistical Evaluation of the ILC

The testability measures are calculated by evaluating the information loss in the diagram of operators, where each operator contributes to this loss. The estimation of the intrinsic losses of information of each operator determines the relevance of the computation of testability measures.

The information loss of an operator is related to the effective information capacity of its outputs. To facilitate the construction of the ITN, we introduce the concept of Information Loss Coefficient, which is the ratio between the effective capacity and the maximum capacity of the output.

If the input domain I and the output domain O of an operator are finite sets, A. Dammak [15] proposes a formula to compute the information capacity of operator.

For the logical operators, the input domain and the output domain are finite sets. The ILC of the logical operators can be evaluated exactly from the truth tables. For the other operators, we propose a statistical evaluation of ILC.

The three principal tasks of the statistical evaluation are:

- Determination of finite data domains;
- Execution of the operator in consideration;
- Computation of the approximate capacity of the operator.

4.1 Determination of finite data domains

Many operators have non-finite input and output domains. For example: the output domain of the

operator *sin* is bounded but continuous [-1, 1]; the output domain of the operator *Rounding* is discrete but not bounded. To evaluate the ILC of these operators, we must have finite data domains (i.e. bounded and discrete).

Information of data domains (lower bound, upper bound, interval) can be found in the specification of the system. In case this information is absent, a study must be done to find the most suitable default domain.

A non-finite domain is transformed into finite domain as follows: if a domain is not bounded, it is necessary to determine a lower bound and an upper bound; if a domain is continuous, it is necessary to subdivide this domain in disjoint intervals. In fact, a non-bounded domain is always limited by MIN_REAL and MAX_REAL, which depend on processors.

4.2 Simulation of operators with random data

The operator in consideration is simulated with random data generated from the finite input domain.

The simulation can be done according to two approaches:

1) A system, which contains the operator, is simulated. The input and output values of the operator are used for the evaluation of the ILC. The advantage of this approach is that one can obtain an ILC value more realistic. But, the singular values can be omitted. And ILC obtained depends on selected architecture.
2) The simulation of the operator is done independently, i.e. this operator is isolated. This approach is relatively simple if the function of this operator and the data domains are known.

The simulation of the operator or the system containing this operator can be done in two ways:

1) We use the simulator of the development environments to simulate the operator (or system) in consideration.
2) We have the code C of the operator (or system), and this code C can be simulated out side of the development environment.

The simulation results are then used to evaluate the ILC value.

4.3 Evaluation of the ILC

Two important factors of the simulation are: the input domains, and the number of executions. In several SIMULINK descriptions, the data domains are not specified. In this case, a research of default domain for each application category should be done.

Suppose that O is the operator in consideration, and Y_O is the output variable of O. The maximum information of its output is $C_{\max}(Y_O)$. The effective information that the operator O produces during the simulation is $C_{sim}(Y_O)$. The ILC value of the operator O is calculated by the following equation:

$$ILC(O) = \frac{C_{sim}(Y_O)}{C_{\max}(Y_O)}$$

With various numbers of executions, we obtain different values of ILC. The purpose of the study on several numbers of executions is: a) to calculate the standard deviation and the average value of ILC; b) to choose a “reasonable” number of executions that is then used systematically for the statistical evaluation of ILC.

4.4 ILC values of some simple operators

The statistical ILC values of some basic operators that we evaluated are represented in the table 1.

Table 1: ILC values of some basic operators

Operator	**Subfunction**	**ILC**
Trigonometry		
	Sin	0.951563
	Cos	0.949281
	Tan	0.466776
	Cosh	0.629261
	Sinh	0.691421
	Tanh	0.683351
	Asin	0.396719
	Acos	0. 397585
	Atan	0.371132
Relational Operator		
	==	0.045415
	~=	0.080793
	<	0.997402
	<=	0.995378
	>	0.995378
	>=	0.997402

These ILC values are integrated into our SATAN libraries.

5. Testability analysis process

In this section, we talk about our testability analysis process. This process was implemented in the MAC tool [16].

Data-flow designs of reactive systems, which are produced by using several development environments (Scade, Sildex, Simulink), are transformed into a common graphical format (dot format): one graph per subsystem of the hierarchical structure.

The hierarchical structure is then flattened into a one-level structure before building the SATAN testability model. In case some operators are not

defined in the SATAN library, the signatures of these operators are used to complete the SATAN library. The ILC value of each operator can be evaluated statistically or manually.

From the elementary ITGs and the flattened diagram, the Satan technology creates the ITG of the design. All flows in the ITG are automatically identified by the SATAN technology. A test strategy is applied to determine the test objectives that are a subset of flows in the ITG.

The ITN of the design is then constructed from the ITG and elementary ITNs. From the ITN and the test objectives, the controllability measure and the observability measure of each functional module in the ITG are computed by simulating the information transfer.

The testability analysis results are represented in a HTML report that supports hierarchical navigation. This report contains a hierarchical table of the testability measurements and labeled operator diagrams. Each labeled operator diagram corresponds to an operator diagram of the system. In this report, users can easily find individual result for each component, and global result for each sub-system of the operator diagram.

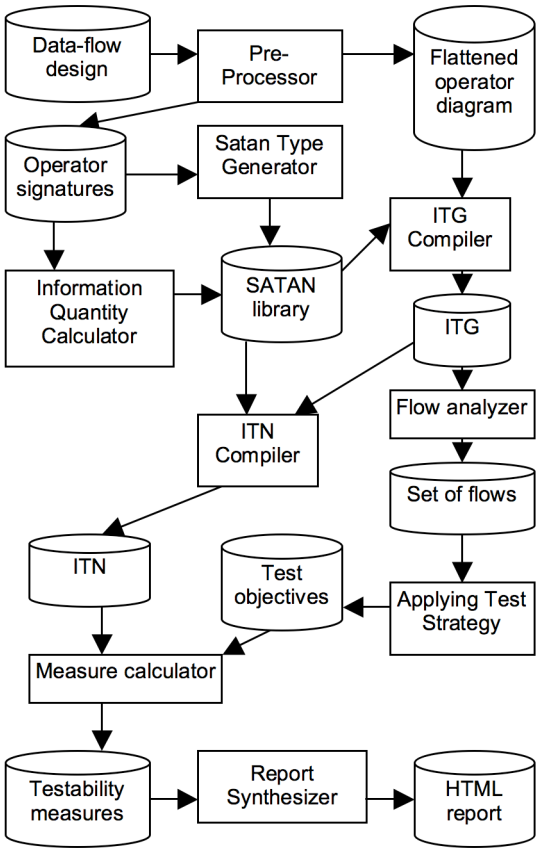

Figure 11: The testability analysis process

6. Case Study

We apply our approach to the top level of the GRS example. Testability measures are presented in Table 2.

Table 2: Testability measures of the GRS example

	Controllability	Observability
Abs	0.9139	0.1246
Comp	0.4978	1.0
Model	1.0	0.8912
Product	1.0	0.8827
Sign	0.9139	0.8954
Sw	0.9704	1.0
sw1	1.0	1.0

Three operators *model*, *Product* and *sw1* have the ideal value of controllability equal to 1. We can say that these three operators are totally controllable. The operator *comp* has the minimum value of controllability equal to 0.4978. It means that in the worst case, only 49.78% of its input domain can be controlled from a flow.

Two operators *sw* and *sw1* are totally observable, because their outputs are directly connected to the two outputs of the system. The observability value of the *comp* operator is equal to 1, because the capacity of its output is too small compared with the capacity of the two outputs of the system. The *Abs* operator has the minimum value of observability equal to 0.1246. This very week value is due to the fact that the output of the *Abs* operator is hidden by the Boolean output of the *comp* operator. In order to increase the observability of the *Abs* operator, the designer can add an observation point to observe the output of this operator at the output level of the system.

We also validate our method on five industrial examples provided by the MBDA Company. These examples were produced in the Simulink and Scade environments. After applying our method, the numbers of test objectives of these examples are considerably reduced: for the biggest example, this number is reduced from 1039 to 41.

7. Conclusion

Our approach allows an automated testability analysis of graphical data-flow designs of reactive systems. This testability analysis, which is applied very soon at the specification phase, helps the designer to identify parts of system that are difficult for testing. Designer can use testability measures to compare different architectures. The test generation is minimized through strategies while ensuring that all operators in the design are covered. Hence, the

validation cost can be reduced. Testability measures can also be used as an indicator in testing resource allocation: tester should pay more attention to parts of system that have low testability measures.

At the moment, we design an “intelligent” filtering to help the engineer in the exploitation of the analysis results with a coding of colors (red and yellow). The colors will give the indication of a relativeness excess of complexity. In the future, we tend to integrate our method into graphic development environments, such as OSATE [17] plug-in in the Eclipse platform, to help developers in reducing the time and the cost of the validation, and in enhancing the confidence of systems.

8. Acknowledgement

The authors thank the Linbox Company (A. Laprevote and C. Delfosse) for their industrialization tasks and helpful advices. The Linbox Company is the editor of the interface wrappers technologies with the Satan technology and the report synthesizer in HTML language.

9. References

[1] http://www.mathworks.com.

[2] http://www.scicos.org.

[3] D. Harel, “Statecharts: A Visual Formalism for Complex Systems”, Sci. Computer Prog., July 1987, pp. 231-274; also see Tech. Report CS84-05, The Weizmann Inst. Of Science, Rehovot, Israel, 1984.

[4] http://www.uml.org.

[5] Peter Feiler, Ed Colbert: "The SAE AADL Standard -An Architecture Analysis & Design Language for Embedded Real-Time Systems", Model-Integrated Computing Workshop, Arlington, VA, USA, October 12-15, 2004.

[6] N.Halbwachs, P. Caspi, P. Raymond and D. Pilaud: “The synchronous dataflow programing language LUSTRE”, Proceedings of the IEEE, 79(9): 1305-1320, September 1991.

[7] P. Baufreton, H. Granier, J.-S. Cruz, F. Dupond: “Visual Notations Bases On Synchronous Languages for Dynamic Validation of GALS Systems“, CCCT’04, Austin, USA, August 14-17, 2004.

[8] H. G. Chalé-Góngora, P. Baufreton, D. Goshen-Meskin, J.-S. Cruz, F. Dupont, R. Leviathan, M. Segelken, K. Winkelmann, N. Halbwachs: “Safeair II Project Advanced Systems development Environment: A methodology and a tool-set designed to develop aeronautics, automotive and space safety-critical systems”,CONVERGENCE’04, Detroit Mi, USA, October 18-20, 2004.

[9] C. Robach and P. Wodey: “Linking design and test tools: an implementation”, IEEE Transactions on Industrial Electronics, vol. 36, pp. 286-295, 1989.

[10] David J.C. Mackay, “Information Theory, Inference, and Learning Algorithms”, Cambridge University Press, 2003.

[11] A. Benveniste, G. Berry: “The Synchronous Approach to Reactive and Real-Time Systems”, Proceedings of the IEEE, 79(9), 1991.

[12] Y. Le Traon and C. Robach: “Testability measurements for data flow designs”, International Software Metrics Symposium, Albuquerque (New Mexico-USA), pp. 91-98, November 5-7, 1997.

[13] H.V. Do, M. Delaunay, C. Robach, J.-S. Cruz, “A Testability Analysis for Data-Flow Designs of Reactive Real-Time Systems”, Proceedings of the Eighth IASTED International Conference on Software Engineering and Applications, pages 318-323, MIT, Cambridge, USA, November 9-11, 2004.

[14] C. Robach: “Test et testabilité de système informatiques”, PhD Thesis, 1979.

[15] A. Dammak: “Etude de Mesures de Testabilité de Systèmes Logiques”, PhD Thesis, 1985.

[16] H. V. Do, M. Delaunay, C. Robach, J.-S. Cruz: “MaC : A Testability Analysis Tool for Reactive Real-Time Systems”, ERCIM News No.58, Special theme: Automated Software Engineering, July 2004.

[17] The SEI AADL Team: “An Extensible Open Source AADL Tool Environment (OSATE)”, Release 1.0, May 23, 2005.

10. Glossary

ILC: Information Loss Coefficient

ITG: Information Transfer Graph

ITN: Information Transfer Net

SATAN: System’s Automatic Testability ANalysis