Determining Configuration Probabilities of Safety-Critical Adaptive Systems

Rasmus Adler, Marc Förster, Mario Trapp
Fraunhofer-Institut für Experimentelles Software-Engineering (IESE), Germany
{rasmus.adler, marc.foerster, mario.trapp}@iese.fraunhofer.de

Abstract

This article presents a novel technique to calculate the probability that an adaptive system assumes a configuration. An important application area of dynamic adaptation is the cost-efficient development of dependable embedded systems. Dynamic adaptation exploits implicitly available redundancy, reducing the need for hardware redundancy, to make systems more available, reliable, survivable and, ultimately, more safe. Knowledge of configuration probabilities of a system is an essential requirement for the optimization of safety efforts in development. In perspective, it is also a prerequisite for dependability assessment.

Our approach is based on a modeling language for complex reconfiguration behavior. We transform the adaptation model into a probabilistic target model that combines a compositional fault tree with Markov chains. This hybrid model can be evaluated efficiently using a modified BDD-based algorithm. The approach is currently being implemented in an existing reliability modeling tool.

1 Introduction

In recent years, dynamic adaptation has gained importance for the development of embedded systems. One major application of dynamic adaptation is the cost-efficient development of dependable systems. In the case of errors, dynamic adaptation can ensure that the best possible system functionality is achieved and that critical functions are kept alive (survivability). Exploiting implicitly available redundancy, dynamic adaptation provides a cost-efficient means to keep up functionalities as long as possible without requiring expensive explicit redundancy channels.

The application of dynamic adaptation for safety critical systems requires a particular specification of adaptation behavior at design time (*predetermined* reconfiguration), because *online-determined* reconfiguration is not verifiable and hardly analyzable [1]. An adaptive system may take various different configurations at runtime. Only a few of them are reasonable, and the benefit of a configuration strongly depends on the probability that the system assumes it. Knowing configuration probabilities, designers can preselect a reasonable set of configurations to be actually implemented. Moreover, probabilistic analyses of adaptation behavior enable safety assessment of adaptive systems. Functional failure behavior often depends on the current configuration. Therefore, only if one knows how probable configurations are, the failure probability of an adaptive system can be determined.

This article introduces a new method to perform probabilistic analyses for safety-critical adaptive systems, supporting design optimization as well as safety analyses. We specify how to automatically transform a model of predetermined adaptation, whose underlying modeling technique has been proven in industrial use, into an equivalent model for stochastic analysis.

The article is structured as follows: in section 2, we review related work on the application of fault trees to adaptive systems; section 3 gives an introduction to the MARS (*Methodologies and architectures for runtime adaptive embedded systems*) modeling concept and deduces requirements for a probabilistic model that can be used as target for the transformation of MARS models; section 4 describes the special, hybrid type of fault tree (FT) we have chosen as target model, as well as the evaluation method used for it; section 5 describes the MARS-to-hybrid-CFT transformation process; section 6 concludes our achievement and gives some ideas for further development.

2 Related work

Dynamic adaptation is a collective term for a wide range of approaches, and no commonly accepted terminology for it is defined in literature. The MARS project focuses on *predetermined software reconfiguration* [1]. This means that system configurations are explicitly specified at design time. We pro-

pose the application of a special kind of fault tree analysis (FTA) to MARS models of adaptive systems.

There are two related approaches that have been previously published. Dugan and Assaf [8] describe an FTA approach for a reconfigurable avionic system that uses both hardware and software redundancy to create fallback levels in case of subsystem failures. Spare situations are modeled by state-based, dynamic fault tree (DFT) gates, and the reliability of the system is evaluated using a combination of Markov-chain- and BDD-based techniques. Szabó and Gáspár [9] determine the dependability of adaptive functions by specialized FT logic, differentiating between detected and undetected component failures. In the authors' words, adaptive functions "change their actual function dynamically according to the faulty input or failure status and can provide information about the success of the execution." Formulas are presented to determine failure probabilities in the presence of detected and undetected failures for the AND, OR and K-OUT-OF-N gate.

The mechanism both author teams denote as adaptation is rather a simple form of redundancy management, which means switching to spare units when failure occurs. In contrast to this, we consider a more dynamic type of adaptation that is indispensable for the development of safety-critical and at the same time cost-sensitive systems [1]. Adaptation of this kind can be extremely complex, as the reconfiguration of one component usually requires the adaptation of further components. Such dynamic interdependencies were taken into account for the first time by the CHAMELEON [2] [3] [4] project and its successor, MARS.

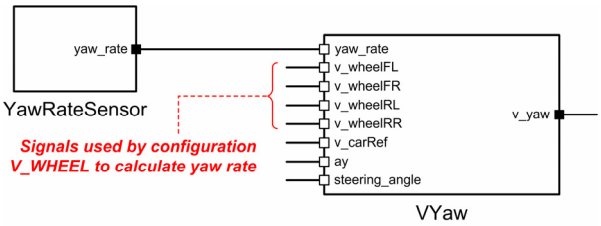

Figure 1. MARS model (partial, high-level view)

We introduce how to automatically transform a MARS adaptation model (Figure 1, example) into an equivalent probabilistic model. The component fault tree (CFT) used as a stochastic equivalent for MARS models was first proposed in [7], motivated by the observation that ordinary FTs do not adequately mirror the compositional structure of today's complex systems. This article extends CFT models by Markov subcomponents to capture multimodal failure behavior of hardware interfaces. Markov processes constitute the stochastic input of the FT, making it possible to calculate configuration probabilities by Boolean logic. As a long-term objective, we aim at safety analysis for systems with complex adaptation behavior.

3 Modeling adaptation with MARS

This section introduces the MARS modeling language used for modeling the behavior of adaptive systems. System architecture is defined using the MARS language, which is an extension of established concepts of architecture description languages [5].

3.1 Modeling concepts

In describing an architecture, we distinguish between *MARS modules* and *MARS components*. *Modules* are the atomic blocks containing the actual functionality specifications. *Components* are structural containers, containing further components or modules. The interface of modules and components is defined by *ports*. Ports are connected via *connections*. Modules communicate with each other using *signals*.

A major problem in specifying adaptation behavior are the cascading interdependencies between modules. Thus, a modular definition of adaptation behavior is necessary to be able to manage the enormous complexity. We achieve this modularity by establishing a quality flow in the system. Besides its actual data, each signal has an additional quality description. Signals are typed by DATIVES (*Extended data type for adaptive systems*) that do not only specify the data values a signal may take but also how the quality of the data can be described. Thus, module designers are able to define adaptation behavior based solely on the quality descriptions available at the local interface. Additionally, they define how the current quality of the provided signals is determined. In this way, it is not necessary to know anything about the rest of the system, and a modular design is made possible.

3.2 Quality description

In our approach all signals are typed by datives. A dative consists of a standard data type, and a quality type. It is important to know the requirements a value must meet, because usually several variants exist to determine the value of a variable. Each variant has different characteristics, which leads to semantical deviations. For instance, if the yaw rate sensor has failed, a car's anti-lock brake system (ABS) can be reconfigured to approximate the velocity of the yaw angle, *v_yaw*, from the speed of the car in longitudinal direction, *v_carRef*, and the lateral acceleration *ay*. The semantics of this approximation is not equivalent to the re-

quired semantics of *v_yaw*. Particularly, if the car is in steep turn and *ay* is influenced by gravitation, the approximated value may significantly deviate from the actual yaw rate. Documenting these semantic deviations is a major aspect of a quality description. Therefore, equivalence classes of variants are represented by *modes*, documenting the deficiencies of the variant. In the first place, the quality type is defined by a set of possible modes QT = $\{M_1, \ldots, M_n\}$. Developers who use a signal know the deficiencies associated with a certain mode and can decide whether or not, respectively how, the module must be adapted in order to compensate for these deficiencies. A *signal s* is an instance of a dative. It has a current data value and a current quality.

3.3 MARS Modules

Based on datives, developers can locally define the adaptation behavior of single modules. Obviously, the module interface has to be extended with respect to the extension of normal data types to datives. Furthermore, behavior specification is no longer directly assigned to modules. Modules may implement different behavior variants, so several configurations can be assigned to a module, each of them representing a behavior variant.

MARS Module interface. In addition to a conventional input/output interface, it is necessary to define a required/provided interface. It defines the direction of quality flow, which is not always identical to the direction of data flow. While data flows from an outport of one module to an inport of another module, quality flows from a provided port to a required port. MARS modules define a mapping of the required qualities of incoming signals to the provided qualities of outgoing signals. If *s.Dative* denotes the dative of a signal, S_R the set of required signals and S_P the set of provided signals of a module, the quality domain is defined by $Q_D = \underset{s \in S_R}{\times} s.Dative.QT$ and the quality codomain by $Q_C = \underset{s \in S_P}{\times} s.Dative.QT$. We call an element from QD *quality input vector* and an element from QC *quality output vector*.

Configurations. A module contains various configurations, each of them representing one behavior variant. It is defined by its interface and a set of configurations. For instance, the module *VYaw* (Figure 1) determines the current yaw rate of the car. This module has several configurations, each referencing one variant for yaw rate determination, and one of them, *V_WHEEL*, uses all four wheel speeds to calculate the yaw rate.

Configurations are defined by the following elements: (1) a guard that specifies under which conditions the configuration can be activated, (2) a priority, (3) an influence definition, defining how the quality of the provided signals is determined. With respect to adaptation behavior, the three latter aspects are of importance and will be looked at in more detail in the following. A guard is a mapping from QD to a Boolean value. The guard defines which signals are required in which modes and is evaluated at run time. It ensures that the quality requirements of the configuration's functional behavior are met. Often, guards of several configurations can be fulfilled at the same time. Therefore, a strict total order is defined on the configurations of a model. From all configurations whose guard is fulfilled, the one with priority, determined by the ordering relation, becomes active. To close the quality flow, it is additionally necessary to define a quality influence (mapping from QD to QC), describing how the quality of the provided signals is determined. For this purpose, an influence is set up for each configuration. An influence contains a list of influence rules. Each influence rule has an influence rule guard that refines the configuration guard and specifies under which conditions the influence rule assigns the current mode to each provided signal.

System. Adaptation behavior is locally defined by MARS modules. Thus, it is possible to specify the system architecture quite straightforwardly. The definition of a system is very similar to nonadaptive systems. It is merely necessary to instantiate modules and to connect those ports of modules and components that propagate the same signals.

3.4 Requirements for a probabilistic equivalent for MARS models

To facilitate transformation. the probabilistic target model should exhibit similar properties as MARS models. The purpose of probabilistic evaluation is to determine probabilities of MARS module or system configurations. The MARS adaptation model is deterministic, meaning that a module configuration at any time depends on the actual state of its environment only. This enables us to exploit a combinatorial model as far as possible.

Besides being at least in part combinatorial, the target model we are looking for should fulfill two requirements in particular. First, MARS models are compositional/hierarchical, so the target model should also be compositional. This way, MARS components can be transformed separately and aggregated step by

step to yield the complete system model. Second, in MARS models, hardware component (sensor/actuator) failures are the source of adaptational evolution; these failures can be transient, and components may exhibit degrading behavior. Thus, the target model should enable the modeling of multimodal, restorable basic "events" (subcomponents).

Although the expressive power of FTs is, in principle, sufficient for probabilistic assessment at this point, they have a couple of deficiencies for our purposes: classical FTs are not compositional and feature only bimodal, nonrestorable basic events. We opt for a combination of a compositional FT model with Markov chains as source of the stochastic process, namely, the usage of CFTs [7] with Markov basic subcomponents.

4 Hybrid component fault trees

Traditional FTs lack an appropriate component concept that would allow the assignment of a separate subgraph to each technical component in a simple way. To overcome this drawback and to facilitate division of labor and reusability of partial FT models, the concept of component fault trees has been proposed. CFTs feature a decomposition approach that is used in a similar way in modern software design notations: subcomponents appear as "black boxes" on the next-higher level and are connected to the environment via their interface (input or output ports). These ports represent the flow of information into and out of components on different hierarchy levels. Similar to a MARS model, CFT models can be instantiated several times in other models; in their environment, instantiations are called "subcomponents". In- and outports of a model appear as subcomponent ports when they are instantiated on the next-higher hierarchy level.

The CFT graph as a whole is defined by its edge connections and the mappings from subcomponents to the corresponding CFT model describing their "internals". The complete CFT can be evaluated stochastically like a classical FT (see [7] for a formal specification of CFTs, [10] for a description of the BDD-based and compositional evaluation algorithm used for CFTs; UWG3, a tool for modelling and evaluation of CFTs, can be found at [11]).

4.1. Combining CFTs with Markov chains

As stated before, CFTs alone are not sufficient as stochastic equivalent for MARS models because FTs neither offer the possibility to express multimodal components nor do their basic events generally allow the modeling of restoration to operational state. For the modeling of sensuator failure behaviour we therefore use Markov chains.

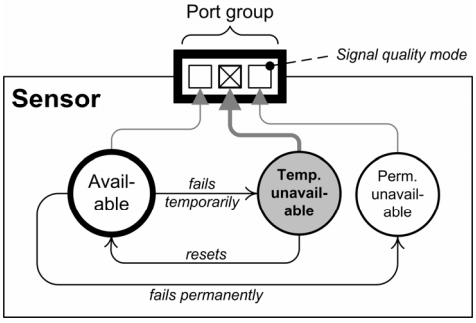

Figure 2. Sensor model with transient failure

In Markov models (Figure 2), a bold circle denotes the starting state (*available*). Two different kinds of edges are used: *temporal edges* with thin arrowhead represent transitions between Markov states, *causal edges* with filled arrowhead, the same as in CFT components, symbolize the flow of state information (Boolean or probabilistic) through a model. The sensor shown is in the state *Temporarily unavailable*, so the associated model outport is set to *true*. The current signal quality mode vector exported by the outport group of the sensor model is, thus, (*false*, *true*, *false*).

4.2 Modeling sensor and actuator failures

To meet the second requirement, the demand for multimodal components with restoration, we introduce general Markov chains as subcomponents in CFT models, which results in *hybrid* CFTs. Using Markov components, the stochastic process of setting the quality input of a MARS model can be expressed, even with an arbitrary number of modes. Markov chains, or their states, respectively, replace basic events in the CFT model transformed from MARS. They are used in the calculation of configuration probabilities in the same way as basic events are used in classical FTA for assessment of system reliability (there are, though, some differences with respect to calculations, see below).

Figure 2 shows the model for a sensor with three states: it can be working properly (state *Available*), it can experience a transient failure and enter the state *Temporarily unavailable*, after which it eventually resets to the working state, and it can fail permanently (state *Permanently unavailable*). All three states (and their associated probability, resulting from the specified failure rates) are exported to the environment via component outports. Other sensors and actuators of the system can be modeled in a similar fashion.

The configuration of a MARS adaptation model is deterministic and depends, at any time, only on the actual quality input into the system. This means the vector of input modes unambiguously determines the component or system configuration, so a combinatorial model like CFT can be used to analyze adaptation. In the translation of a MARS model to a hybrid CFT, (state-based) Markov components represent the quality modes of the system's hardware interfaces it adapts to internally, while (combinatorial) CFT components specify adaptation behavior; they process Markov input, or input from other CFT components.

4.3 Evaluation of hybrid CFT models

UWG3, the industry-proven tool we are using to create and evaluate CFT models, has been reengineered for the integration of different model types and renamed to ESSAREL [11]. Like UWG3, ESSAREL uses a special, compositional BDD algorithm for CFT evaluation [10], tailored to their compositional structure. In hybrid CFTs, Markov states take the place of basic events and are converted like them to BDD variables during calculations. Nevertheless, a couple of semantical differences have to be taken into account in the algorithm. Of special interest is the export of several states of the same Markov component (and their associated probability) to the CFT environment, as is the case in hybrid CFTs resulting from transformation from MARS models. Since a component can be in only one state at a time, states of the same Markov component are mutually exclusive.

The traditional BDD-based algorithm for FTA assumes state probabilities (that is, basic events) to be stochastically independent, so BDD evaluation has to be adapted to treat variables representing mutually exclusive states correctly. This can be done using a special approach developed in [12] that has been implemented in ESSAREL.

5 Transformation of MARS models

MARS models describe the adaptation behavior of adaptive systems based on the quality flow established in the system. The qualities provided by sensors and actuators are taken for granted. With the help of quality flow, reconfiguration of the system results automatically from these sensuator qualities.

A similar approach can be used for determining the probability that a certain configuration of a MARS module is active. Sensuator qualities are described by Markov chains that define the basic quality input of the system. The quality flow can then be expressed by a purely combinatorial model. The multimodal quality concept in MARS suggests the use of multivalued logic for representation, but since our CFT target model is Boolean, we use Boolean functions instead of multivalued ones. For each signal quality mode we define a Boolean variable. When a signal is in a certain mode, the corresponding variable has the value *true*.

5.1 Transformation of a MARS module

Figure 3 shows the lower part of the CFT structure that results from the transformation of a non-sensuator MARS module to a CFT component. In the following we call such a CFT component *Prime component* (PC). MARS modules define a mapping M from QD to QC. We will show in the following that the underlying function F of the prime component defines a mapping that is equivalent to M. F: CFT_QD $\rightarrow$ CFT_QC where CFT_QD is the binary encoding of MARS QD, CFT_QC is the binary encoding of MARS QC. Transformation of a MARS module into a CFT is done in four steps, by transforming (1) configuration guards, (2) configuration guard priority, (3) configuration influences, (3.a) influence rule guards, (3.b) influence rule guard priority, (4) the mapping of influences to their associated quality output vectors.

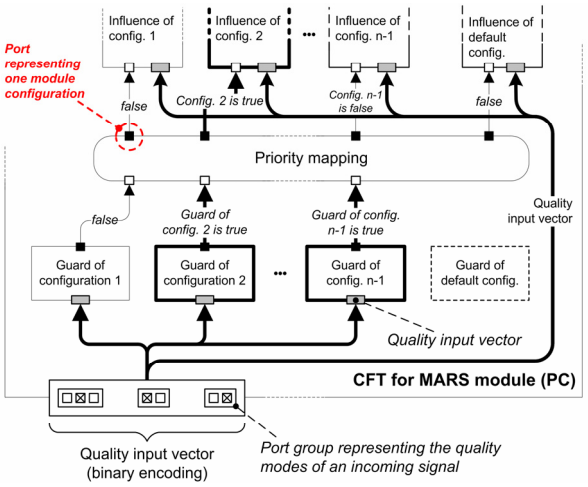

Figure 3. Main part of a Prime component

In CFT diagrams, single ports and quality vectors/port groups are distinguished between having direction *in* (white fill) and *out* (black fill). Ports of a port group have a white fill always and denote their Boolean value by a cross (*true*) or no cross (*false*).

Like a MARS module, its CFT transformation maps quality input vectors to quality output vectors. Configuration guards evaluate the current input vector and

map it to a Boolean variable. Since more than one configuration guard may yield a *true* value, there is a ranking among configurations (given by their strict order in MARS) that will result in the selection of exactly one configuration to be *true* (that is, active). The single active configuration determines its associated influence, which determines the actual value of the quality output vector via evaluation of its influence rule guards and their strict order.

Transformation of guards. CFT configuration guards and influence rule guards are Boolean functions, or fault trees, respectively, that map the quality input vector to a single Boolean value (Figure 4).

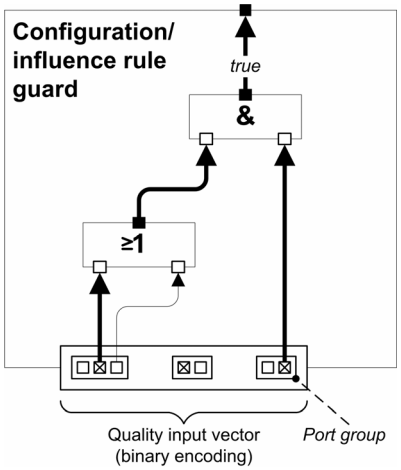

Figure 4. Guard transformation (example)

Note that a guard may evaluate any nonempty subset of elements of the input vector, as shown in Figure 4. While it only makes sense to combine variables from the same port group by *disjunction*, since they represent mutually exclusive quality modes of the same signal, quality modes of different signals may be combined by either disjunction or conjunction. The evaluation of a guard function to *true* is necessary (but not sufficient) for the corresponding configuration or influence rule to become active.

Transformation of priority. Of every configuration (*C*) or influence rule (*IR*) of a configuration whose guard is fulfilled, the one with priority becomes active. Priority is derived from the strict order in MARS:

$$>_p \subseteq \{F_{C/IR_1}, \cdots, F_{C/IR_n}\}^2, F_{C/IRi} >_p F_{C/IRj} \Leftrightarrow$$

Configuration/influence rule *i* has priority over configuration/influence rule *j*.

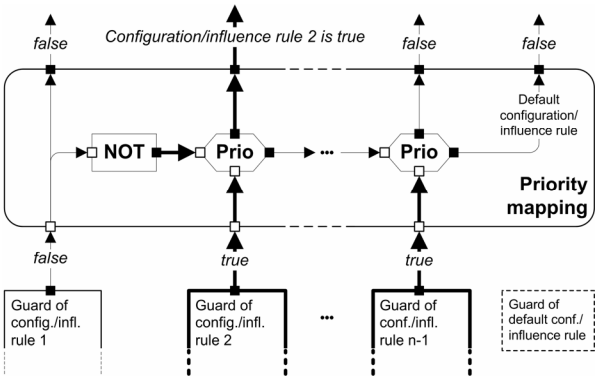

Figure 5. Transformation of priority

This means if, and only if, a guard is fulfilled *and* all guards of higher-ranking configurations/influence rules are not fulfilled, the associated configuration/influence rule becomes active. Strict order in MARS is translated to a generic CFT component *Priority mapping* (PM). From all guards that yield *true*, it singles out the guard with priority, resulting in selection of the active configuration/influence rule (Figure 5).

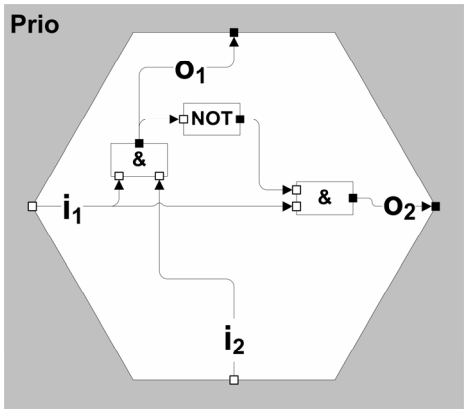

Figure 6. Composite gate *Prio*

Note that the guard of the default configuration or influence rule does not need to be modeled since it is always *true* (in figures it is therefore shown with a dashed border). The default output of priority mapping is *true* if and only if no modeled guard evaluates to *true*. Figure 5 shows the use of the PM component in transformation. Although both *guard 2* and *guard n-1* evaluate to *true*, only *configuration/influence rule 2* is set active because it has priority.

To model rank order, a special composite "gate" *Prio* is used (Figure 6). This gate has two inports (i_1, i_2) and two outports (o_1, o_2). Inport i_1 is *true* when no higher guard is *true* and otherwise *false*. Inport i_2 is

true when the guard connected to it is fulfilled, otherwise *false*. Outport o_1 denotes the activation of the configuration/influence rule associated with the guard (*true* = active; *false* = inactive). Outport o_2 propagates the information whether this and all preceding, higher-ranking guards are *false*.

Transformation of influences. An influence component consists of a number of influence rule guards that are, like configuration guards, strictly and totally ordered (Figure 7). For this reason, the structure of an influence transformed to a CFT component resembles the module structure shown in Figure 3. There is exactly one outport of the PM component that has a *true* value, representing the active influence rule. As with configurations, the active influence rule is the one associated with the influence rule guard that has priority.

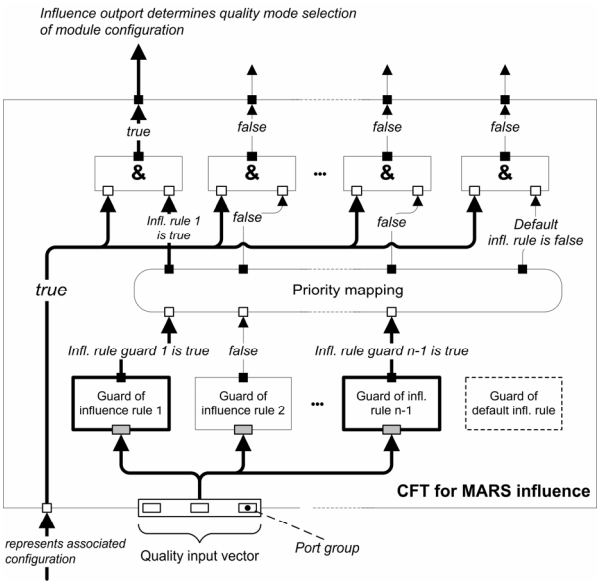

Figure 7. Influence transformation

Besides the quality input vector, every influence has an additional single Boolean inport, representing its associated configuration. Since an influence should only have an effect if its configuration is also active, every outport of the PM component is combined with the Boolean inport using an AND gate. Thus, exactly one influence outport is *true* when the associated configuration is active, otherwise, all outports of this influence are *false*.

Determination of quality output vectors. The transformation of a MARS module described so far ensures that for each possible quality input vector exactly one influence outport will be *true*.

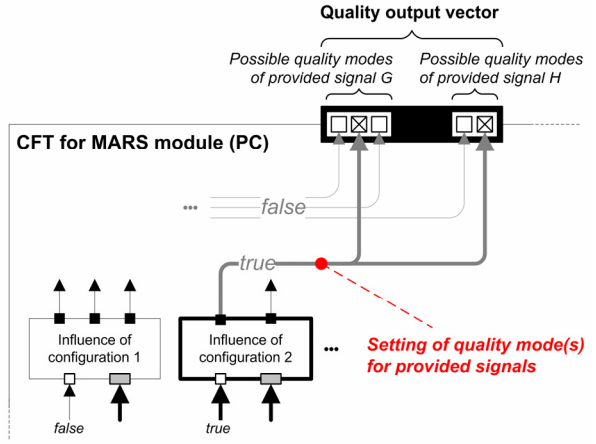

Figure 8. Mapping to quality output vector

Every influence outport is connected to one port in each prime component outport group, and this means that the combination of signal qualities that constitutes the quality output vector is set (Figure 8, showing the missing top part of Figure 3). Since all other influence outports will be *false* and every PC outport is target for at least one influence outport, the value of the quality output vector is well determined at any time.

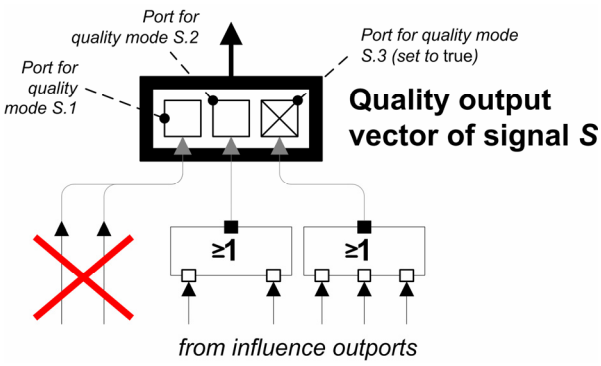

Figure 9. Several influences setting the same mode

Note that ports may only be directly targetted by one edge, so if more than one influence can set an outport to *true*, incoming edges are combined using an OR gate (Figure 9).

Resulting global structure. After transforming all necessary MARS modules, deriving the complete transformation of the MARS model is very intuitive. Corresponding PC in- and outports denote the same quality mode of the same signal and simply have to be connected with each other according to the system structure (see Figure 1 for an example). They are also connected to the Markov basic subcomponents that model sensuator behavior on which the system's adap-

tation depends. For every configuration of a MARS module the probability of becoming active can then be calculated by FTA and sampled at the corresponding PM outport of a PC (Figure 3).

5.3 Benefits of the probabilistic model

Based on a quantitative evaluation of the hybrid CFT, early design decisions concerning the implementation of a configuration can be made. Take as example a MARS module specifying ABS functionality with degradation levels realized in four configurations: (1) ABS with yaw rate correction and automatic steering control has highest priority and an average probability of 90% during mission time, (2) ABS with yaw rate correction and without automatic steering control has the second highest priority and an average probability of 0.0001%, (3) ABS without yaw rate correction has the third highest priority and an average probability of 8.9999%, (4) No ABS available has lowest priority and an average probability of 1%.

The added value of the second-highest configuration in comparison to the third-highest is small. Considering the low probability that it becomes active, the additional effort to implement it is not justified. Furthermore, decisions concerning system safety are also possible. Using quantitative analysis, system architects can make sure that the probability of an undesirable configuration, such as "No ABS available", is below a certain threshold. Finally, the proposed method can be exploited for improved dependability assessment by combining it with functional failure models. For each PM outport of the prime component of interest, representing one configuration probability, a functional failure model is created. By conjunctions, failure probabilities are weighted with configuration probabilities to yield a more realistic failure probability of the system. This provides the basis for advanced safety analyses of adaptive systems.

6 Conclusion and future work

Cost-efficient development of safety-critical adaptive systems requires the specification of possible system configurations during design phase. Over several years, the adaptation modeling concept of MARS has been applied in various case studies in the industrial development of vehicle dynamics control systems. The contribution of this article is a technique for transforming MARS models into hybrid CFT models, which makes it possible to calculate the probability of being active for each MARS configuration. On this basis, crucial information for design decisions is provided and so the development effort spent on the desired safety level of adaptive systems can be optimized. Furthermore, we point out a way to perform safety analyses of these systems.

The MARS-to-hybrid-CFT transformation is currently being implemented in the ESSAREL tool [11]. As future work, we plan to define a process for the integration of functional failure models with the probabilistic adaptation model. MARS models can also contain cyclic dependencies. As CFTs cannot have cycles, we will develop a way to integrate cycles by introducing timing, which can be done using SEFTs [13] instead of CFTs. This will make it possible to analyze realistic, industry-sized adaptation models.

7 References

[1] Trapp, M, R Adler, M Förster, J Junger (2007). Runtime Adaptation for Safety-Critical Automotive Systems. *IASTED Software Engineering, Proceedings* (552), Innsbruck, Austria, 2007

[2] Trapp, M (2005). Modeling the Adaptation Behavior of Adaptive Embedded Systems. *Doctoral dissertation, TU Kaiserslautern*.

[3] Trapp, M, B Schürmann. On the modeling of adaptive systems. In *International Workshop on Dependable Embedded Systems*, Florence, 2003

[4] Trapp, M, B Schürmann, T Tetteroo. Service-based development of dynamically reconfiguring embedded systems, *IASTED Intern. Multi-Conference on Applied Informatics*, pp 935–941, Innsbruck, 2003.

[5] Medvidovic, N, R N Taylor (2000). A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Trans. Software Eng.*, 26(1).

[6] GME: http://www.isis.vanderbilt.edu/projects/gme/.

[7] Kaiser, B, P Liggesmeyer, O Mäckel (2003). A new component concept for fault trees. *Proceedings of the 8th Australian workshop on safety-critical systems and software* (SCS'03). P Lindsay & T Cant, Eds.

[8] Dugan, J B, T S Assaf (2001). Dynamic fault tree analysis of a reconfigurable software system. *19th Intern. system safety conference*. Huntsville, Alabama, September 2001.

[9] Szabó, G, P Gáspár (1999). Practical treatment methods for adaptive components in the fault tree analysis. *1999 Proc. Annual reliability & maintainability symposium*, pp 97-104.

[10] Kaiser, B, A Zocher (2005). BDD complexity reduction by component fault trees. *ESREL 2005, Gdansk*.

[11] ESSaRel website: www.essarel.de.

[12] Zang, X, D Wang, H Sun, K S Trivedi (2003). A BDD-based algorithm for analysis of multistate systems with multistate components. *IEEE Transactions on computers,* 52: pp 1608-1618.

[13] Kaiser, B (2005). State/Event Fault Trees: A Safety and Reliability Analysis Technique for Software-Controlled Systems. *Doctoral dissertation, TU Kaiserslautern*.