Appeared in: Electronic Systems for vehicles, VDI Verlag, Duesseldorf, 2001. p.53 - 73

Integration of Design and Diagnosis into a Common Process

Integration von Design und Diagnoseerstellung in einen gemeinsamen Prozeß

Dipl. Ing. **R. Brignolo,** Magneti-Marelli SpA; Dr. **F. Cascio,** Centro Ricerche Fiat; Torino; Prof. Dr. **L. Console**, Universita di Torino; Prof. Dr. **P. Dague**, Université de Paris Nord; Dipl. Ing. **P. Dubois,** Renault, Paris; Dr. **O. Dressler,** OCC'M Software GmbH, München; Dipl. Ing. **D. Millet,** PSA Peugeot Citroën, Paris; Dipl. Ing. **B. Rehfus**, DaimlerChrysler AG, Stuttgart; Prof. Dr. **P. Struss,** Technische Universität München

Zusammenfassung

Die ständig wachsende Bedeutung der Onboard-Diagnose für Kraftfahrzeuge erfordert eine entsprechende Integration der Diagnoseerstellung in den gesamten Designprozess. Dieser Bericht beschreibt die innerhalb des europäischen Projektes „Integrated Design Process for onboard Diagnosis" (IDD) erarbeiteten Basiskonzepte, welche neue Lösungsmethoden zur besseren Integration diagnoserelevanter Entwicklungsschritte, wie z.B. Diagnostizierbarkeitsanalyse, FMEA, Onboard-Diagnose-Erstellung, in den Designprozess mechatronischer Subsysteme aufzeigen.

Abstract

The constant growing importance of onboard diagnosis for automobiles demands a corresponding integration of diagnostic tasks in the entire design process. This report decribes the basic concepts worked out within the European project „Integrated Design Process for onboard Diagnosis" (IDD), which show new solutions for a better integration of diagnosis related tasks, such as diagnosabilty analysis, FMEA, onboard diagnosis design, in the total design process of mechatronic subsystems.

1. Introduction

1.1 Motivation and Goals

The importance of diagnosis in onboard automotive systems is constantly growing together with the complexity of the systems. The average dimension of the diagnostic code inside a modern electronic control unit (ECU) is now more than 50% of the whole code. At present, there is no correspondence between such an important role of diagnosis in onboard systems and a similar role that diagnosis could play in the design process chain.

The correct way of dealing with this situation is *to re-organise the design and development chain so that the diagnosis is no longer the last task in the design chain*, but starts together with system analysis and goes along the whole design process.

1.2 The IDD - Project

The European Fifth Framework project „Integrated Design Process for onboard Diagnosis“ (IDD) pursues the goal to formalise and standardise the diagnostic design process, and to enable the introduction of diagnosis early in the chain. This methodological goal has to be combined with another important objective: *giving to the designers a set of tools that can help them in evaluating and understanding the effects of each choice on the system being designed.* Model-based diagnosis (and, more generally, model-based systems) is the fundamental methodology that supports the objectives of the project.

IDD was started February 2000 with a duration of three years and involves both industrial and academic partners:

- Fiat CRF, Torino
- Magneti-Marelli SpA,
- PSA Peugeot Citroen, Paris
- Renault, Paris
- DaimlerChrysler AG, Stuttgart
- OCC'M Software GmbH, München
- Universita di Torino
- Université de Paris Nord, XIII
- Technische Universität München

This paper describes some basic concepts developed in the first half of the project and outlines the current and future work on the development of model-based software tools supporting design. We started by performing an analysis of the current design process (using as test-benches some departments of our application partners), paying particular attention to the role of diagnosis-related activities (such as diagnosability analysis, FMEA generation, generation of onboard diagnostic software). Then we identified some weaknesses and analysed the consequences that such weaknesses can have on the overall design process (e.g., time delay due to re-design of some parts of the system, poor diagnosability, etc.).

Next, we defined a new design process in which the activities related to diagnosis are integrated into the design loops. The actual implementation of this new process requires new software environments, in which different tools are made available to the designer who can use them while designing a new system to perform different activities such as: diagnosability analysis (i.e., checking if the system is diagnosable, given the current set of sensors or asking the system which additional sensor would allow the discrimination of a given set of faults), "what-if" analysis comparing different design choices (and the consequences of these choices from the diagnostic point of view), generation of the FMEA (this task can be partly supported by model-based systems), and generation of onboard diagnostic software. In other words, these tools can allow the designer to make different types of analysis on the system being designed and then to take more informed decisions.

At the current stage, the project works on the software specification, the software architecture, and the identification of the tools to be integrated in the demonstration platform that we will implement in the next phases of the project.

The results are being applied to two different classes of automotive systems: a classic one with a central ECU and a second one with a distributed architecture composed by different ECUs connected in an onboard network. For a general use of the methodology, the interface between the different tools is based on the XML language.

2. Current Design- and Diagnostic Process

2.1 General Overview

The current processes of each industrial partners have been investigated with a focus on the integration of the diagnostic process and diagnosis-related processes into the whole design process of mechatronic subsystems.

Starting from these results a „merged process“ has been developed that is based on the similarities recognized, ignoring details and small differences. The abstraction of this process will be used as a comprehensive *reference* for the current design processes. First this chapter gives an overview of the general process, then, in the next section, the abstract representation of the reference process.

Four general phases can be recognized in an entire vehicle design process

- Strategy phase
- Technology phase
- Integration phase
- Production phase

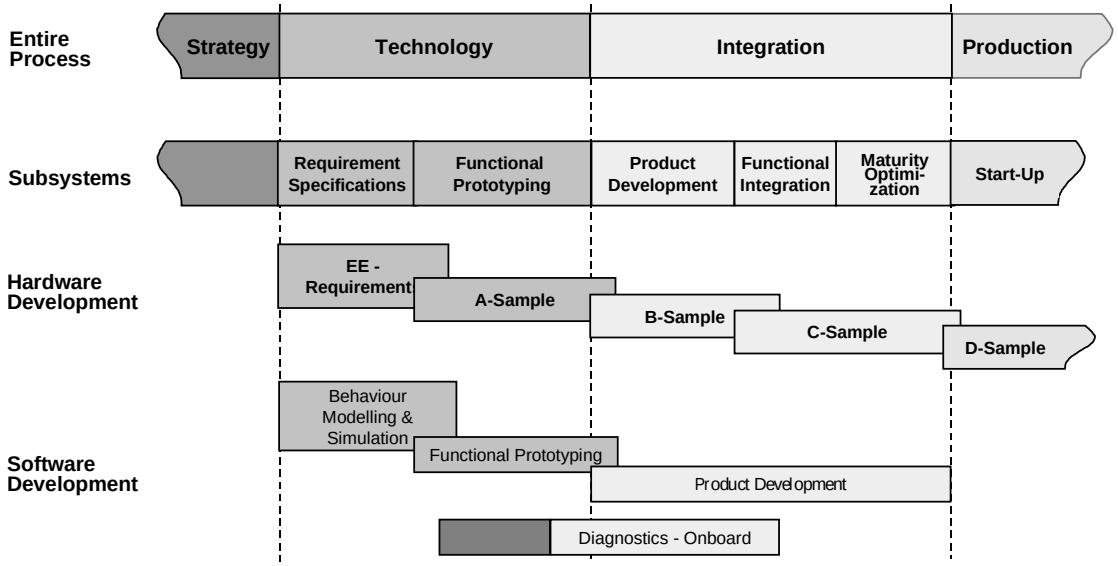

Fig. 2-1: Entire Process and subsystem process, overview

During the ‚strategy phase' a first conceptual framework for the new product is worked out, the ‚technology phase' targets the concept approval, the ‚integration phase' focuses on the realization of the new product by taking into consideration technical feasibility and manufactering aspects, and finally the ‚production phase' ensures the industrial mass production with the correct requirements of quality.

In the framework presented here we consider especially processes related to *mechatronic subsystems*, such as air conditioning or engine control systems. These subsystems involve ECUs as centers of control and diagnostic functions, and the physical system, comprising mechanic, hydraulic, electric components. Following [1], Fig. 2-1 summarizes the overall design, isolating the different phases and showing in which way the process for a subsystem, which is the most interesting one in this project, is related to the entire process.

The IDD approach focuses primarily on the Technology phase which leads to the first almost complete prototype, but takes into account that a good amount of diagnostic development is performed at present in the Integration phase, as illustrated in Fig. 2-1.

2.2 The Reference Process

From an abstract point of view, the reference process, which is focussed on the functional prototyping within the technology phase, can be modelled as a set of nested loops.

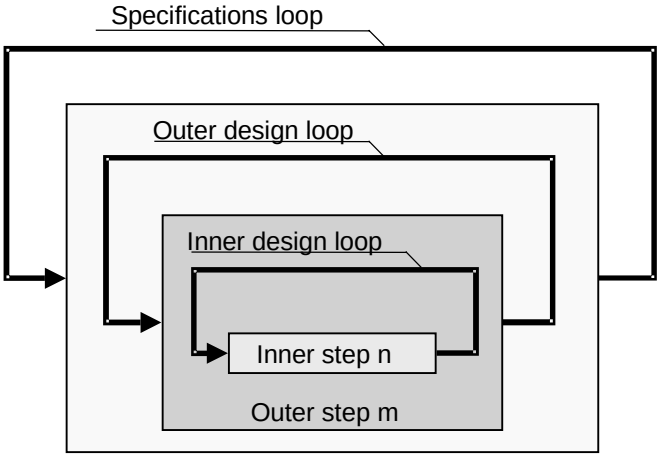

Fig. 2-2: Abstract reference process

In the style of the well known ‚V-model' different design loops have been identified, which are described in more detail as follows.

Specifications loop

Definition of requirements/specifications and implementation of the validated result. In this phase also feedback from after-sales/ customers may be involved. Further requirements may be added depending on mock-up observations.

Outer design loop

Design of the whole system prototype, involving the definition of the overall structure of the system, i.e. the selection of the physical (mechanic, hydraulic, electric) components and decisions about the overall layout of the system. This loop terminates when the prototype meets all the requirements and specifications. The core activities are design of the system including its control and diagnosis, comprising a series of inner design loops, and the hardware development of the physical system, which runs in parallel.

Inner design loop

Design of the ECU-based control system and components. Each iteration involves the design of the control algorithms, FMEA, diagnostic development, implementation of the ECU (HW and SW) and verification of the algorithms, see Fig. 2-3. The verification step at the end of the first iterations is performed using models (software/ hardware in the loop), whereas, later, the physical system is used. Depending on the achieved results, there are several iterations, each one of them producing an advanced prototype.

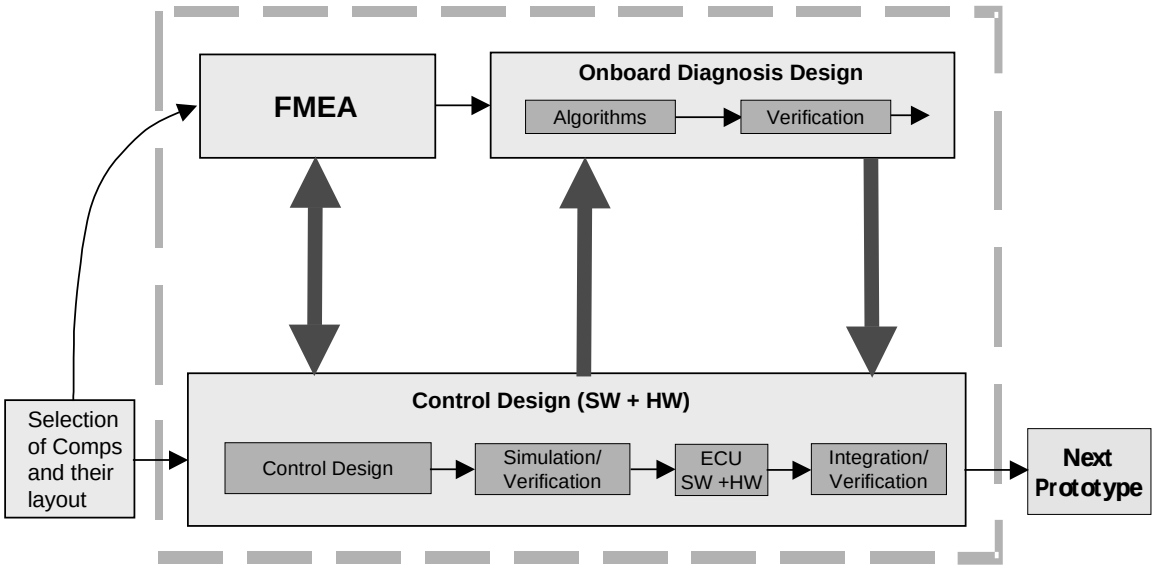

Fig. 2-3: The reference process, one iteration of the inner design loop

3. Definition of a new Process

The next section summarizes the analysis of the reference design process, which aims at identifying how the integration of diagnostic relevant tasks in the overall design process could be improved. Section 3.2 shows how these improvements can be brought together to a new process frame that comes along with a new tools architecture.

3.1 Analysis of the reference process

Three problem areas in the reference design process have been identified as the essential ones with respect to a better integration of the diagnostic tasks, mainly in the inner and the outer design loops.

The first problem area concerns the interaction between the diagnosis design process and the FMEA generation (cf. upper part of Fig. 2-3).

- FMEA and generation of onboard diagnosis are separated and sequential tasks.
- Only few tools support the information extraction process needed for the FMEA, e.g. simulating the consequences of faults or studying interactions between faults. Thus a lot of work is left to the experience and sensibility of the people that perform FMEA.

As an improvement, the two activities could be integrated starting from a *common knowledge base*, which consists of the (correct and faulty) behaviour of the device and the components, preferably expressed by qualitative models. Within a model-based-systems approach most of the common activities of the two tasks could be improved based on a shared model and performed together in a coordinated way.

The benefits would be:

- support for FMEA compilation
- support for diagnosis specification design
- more reliable FMEA
- the FMEA process compilation could be based on a standard procedure
- stronger integration of diagnosis and FMEA (e.g. both include an analysis of fault detectability)
- no necessity to perform the tasks in sequence.

The second problem area concerns the interaction between the development of FMEA and diagnosis, and the development and design of control algorithms of the system (cf. Fig. 2-3).

Currently, these are two substantially separate tasks, that can interact one another. Examples for possible interactions are:

- a change of the control algorithm may turn a physical component, that was not very essential before, into a critical one and, hence require additional diagnostics
- a change of the control algorithm promotes the masking of certain faults that were detectable more easily before. Again, additional diagnostics have to take this into account.
- a change of the diagnostics aiming at enhacing diagnosability may exploit additional signals, which may possibly improve control, as well

As a consequence, requirements and constraints arising from one of these tasks, can be dealt by the other ones only in the next inner design loop, i.e. changes in the design of control algorithms can have impact on FMEA/ diagnosis only during the next inner design loop and vice versa, thus causing additional iterations and time delay.

As an improvement, the control design and the diagnostic process should be integrated in such a way that there is only one single task for the development of the system model combined with an automatic extraction of the diagnostic model and that, based on this model, there are tools for FMEA, diagnosability analysis and diagnosis design, integrated with those used for control design.

This results in a number of benefits:

- direct interaction between the control development and FMEA - diagnosis process that can positively influence each other in finding optimised solutions.
- time saving by a smaller number of iterations during the inner design process
- only one (system) modelling task.

The third problem area concerns the relation between the design of diagnosis and component selection and layout definition (cf. left-hand part of Fig. 2-3).

The problem here is, that currently the component selection task is external to the inner design loop. As a consequence, for instance the choice or placement of sensor is often not optimized with respect to diagnosis purposes, or, if later changes are made, additional (outer and inner) design loops are needed that cause delays.

An improvement could be reached by performing a comparative analysis (‚what-if-analysis') inside the inner design step and the integration in the early phases of control and diagnostic development. Thus, part of the component selection task is moved inside the inner design process, and, in particular in the early phases of the inner design loop, it is possible and cheap to modify component choices, e.g. sensors, regarding type, sensitivity or placement and to immediately explore the impact on control generation, FMEA, diagnosability analysis, and diagnosis generation.

The expected benefits are:

- integration of components selection in the early phases of inner design
- better diagnosability analysis
- possibility of comparing alternative solutions regarding component selection/ placement
- impact on reliability of the resulting system
- time reduction by decreasing of the number of inner/ outer design loops
- more informed decisions about component selection, e.g. about the consequences of changing type or placement of sensors

3.2 The new process

Based on this analysis of the reference process and the outlined improvements, we propose a frame for a new process which is closely connected with a new tool architecture.

In summary, the framework for a new process has to satisfy the requirement that in the inner design loop of the process, the designers (the different experts involved in the design) should be supported in performing different activities in an interleaved way:

- design of the physical system
- design of control algorithms, and their simulation (for quantitative analysis)
- generation of the FMEA of the designed system
- analysis of the diagnosability, i.e. investigation which faults are detectable and discriminable from each other
- derivation of on-board diagnosis (OBD) software for the system
- comparative analysis on the current design (physical system and control), i.e., analysis of the consequences of applying changes to the design both from the control and diagnosability point of view.
- comparative analysis of different design alternatives

Thus, designers and decision makers are supported in the process of evaluating trade-offs between different designs and in making choices about the best design of a system.

This results in the following framework for an new process as shown in Fig. 3-1.

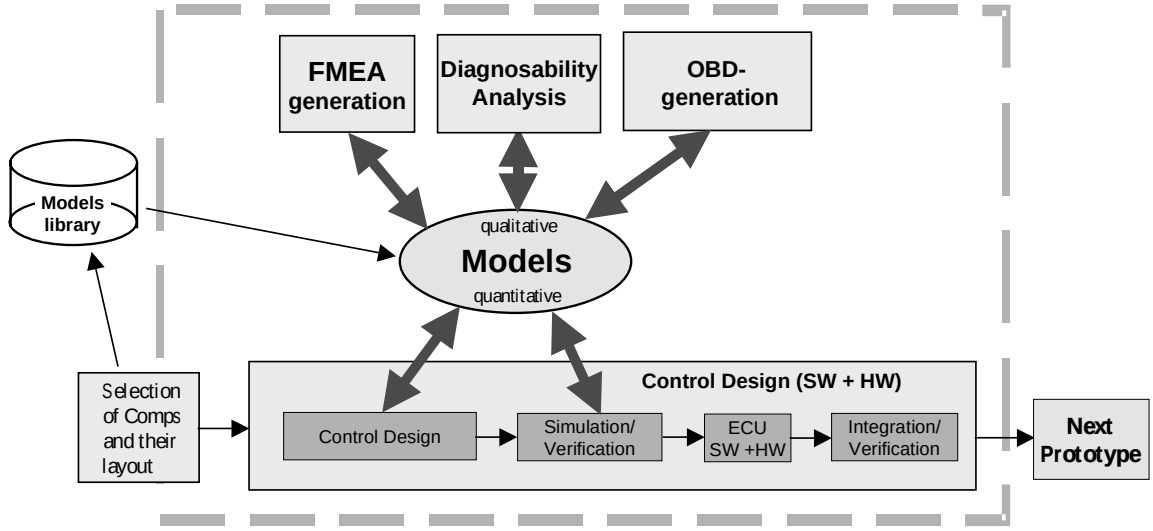

Fig. 3-1: Frame for the new design process

4. Software Support for the New Process Framework

4.1 Model-based Tools Architecture

This proposal has two important consequences, that also affect the relevant software tools:

1. Such a tight integration of different activities and the aim to perform them concurrently require the fast and reliable exchange of information about any changes in the design introduced by any of the activities. This is why we propose that *the ‚**model**‘ of the system being designed must play a **central role** in the new process,* as indicated by the picture*.*

2. The aims to update FMEA, diagnosability analysis and OBD generation quickly after a change and to consider different design alternatives in parallel established the requirement that these tasks can be effectively supported or automated by respective computer tools based on the model, i.e. they have to be model-based tools.

Accordingly, the actual goal is to provide a new set of functionalities for supporting the designer, which are mapped to certain tools that are realized as ‚software plug-ins‘ added to the existing software tools for design. Within the scope of IDD, we are considering three plug-ins:

- tools for diagnosability analysis
- tools for supporting the FMEA generation
- tools for supporting the generation of onboard diagnostic strategies and software

These tools rely on model-based systems and will be based on a common set of models, qualitative diagnostic models, and a common model-based diagnostic system core.

The new solution for the process and tools architecture should be able to be integrated or combined with the simulation tools, that are currently used for the design of control strategies, like Matlab-Simulink, based on quantitative models. This requires software that transforms the models created in these environments into qualitative diagnostic models that form the basis for the model-based tools.

Fig. 4-1 summarizes the overall architecture of the new design support system .

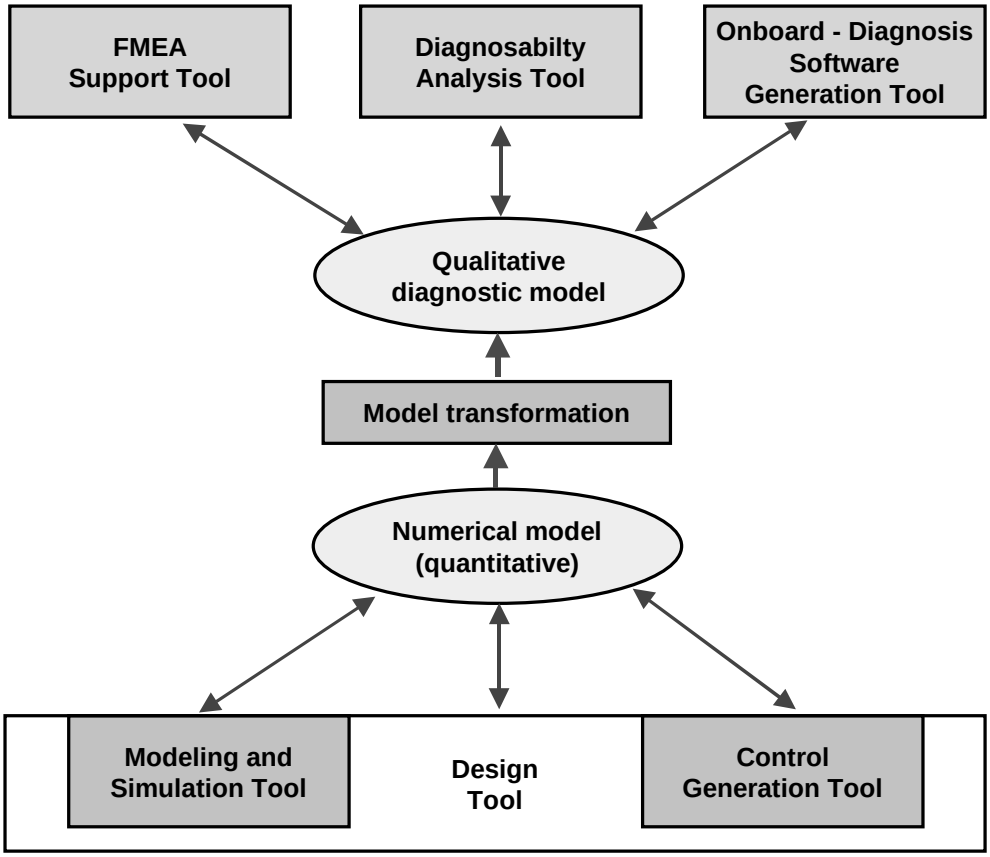

Fig. 4-1: Tools architecture for the new process

The figure shows the different modules and points out the fundamental role of (i) the design tool, to which the other tools are added as new functionalities that can be called during the design process and (ii) the models which form the link between the various tasks to be integrated and which is the basis for the integration. (The module for the generation of control software is outside the scope of the work regarded here.)

Models are on the one hand a universal utilizable models library, comprising the models for normal and faulty behaviour of the components, that are obtained by decomposition of the systems, and on the other hand, the ‚system models', which correspond to the composed models representing structure and behaviour of the specific system under examination.

In the next section, we briefly discuss the principles underlying model-based systems and the benefits expected for the project goal.

4.2 Model-based Systems

Model-based reasoning aims at representing domain knowledge about “first principles” in a declarative and modular way in order to use it for different problem solving tasks. In a technical domain, this comprises engineering knowledge about the behavior of products and their parts, manufacturing processes etc. This has to be regarded in contrast to methods which basically capture empirical associations rather than the underlying physical and engineering principles and rigorous deductions based on them.

Model-based reasoning takes a reductionist's view on systems, attempting to describe complex systems in terms of interactions of more basic ones.

This allows for the separation of the

- *structural description* of a system (its “blueprint”), from the
- *behavioral description* of its constituent elements which are collected in model libraries.

The basic goals of model-based reasoning are then to *predict, analyze, and compare the behavior* of an aggregate system based on the knowledge about its structure and the constituents’ behaviors.
This approach *intrinsically addresses the variant problem.* The structure may vary freely. Behavioral descriptions of constituent element *types* are re-used over and over. A maintenance and development effort only becomes necessary when substantially new and different constituent types occur.

Besides this basic feature of *compositional modeling*, it is often required that a model fragment does not introduce unwarranted details. Describing faulty behavior in too much detail for the purpose of diagnosis or FMEA is an example. Without the ability to describe classes of behavior such model fragments are most certainly useless.
Therefore

- *qualitative modeling* is a key ingredient: the models should make *essential distinctions only*. This enables them to capture the behavior of *classes* of systems in *classes* of situations, thus enhancing re-usability of models across applications and tasks.

Although the goals are quite ambitious and subject to on-going substantial research around the world, a set of principled methods, techniques, and industrial-strength tools have been developed that provide a solid foundation for application. This holds, in particular, for the most advanced field, *model-based diagnosis*. The following section briefly discusses its key steps. These also appear in other tasks such as failure modes and effects analysis and testing which are not presented in detail.
The principle underlying model-based diagnosis exploits model-based behavior prediction by *comparing* the predicted (correct or faulty) behavior with the actual observations.

A model-based diagnosis system essentially works by cycling through the following five subtasks:

1. *Observations* of the actual behavior are entered either manually or automatically, e.g. via an on-line connection.
2. *The aggregate behavior model* computes conclusions about system parameters and variables (observed and unobserved), and thereby predicts the expected behavior.
3. If a *contradiction* is detected (i.e. conflicting conclusions / observations for a parameter or variable) the set of component modules involved in it indicates which components possibly deviate form their intended behavior. This can be determined by the diagnosis system because the aggregate behavior model has structure. It reflects the device constituents that may incorporate faults.
4. Solution candidates are generated. They are called candidates because they must still withstand the consistency test in the next cycle to be true solutions. In case models of faulty components' behavior are provided, the same approach (checking consistency of a model with the observations) can be used to discard particular faults to exonerate components based on the tentative assumption that the set of fault models is complete or that an unknown fault is unlikely.
5. Suggestions for useful *probes and tests* are generated. This is possible because the behavior model reveals where the diagnosis candidates entail distinctive features of behavior.

These key steps are fairly basic, simple, and straightforward. Nevertheless, they provide the foundation for diagnosis and other systems that address several key problems, especially problems arising in industrial contexts.

- Diagnosing *new devices* becomes possible without available experience with them. This holds because and as long as they are assembled from components whose behavior models are contained in a model library.
- Unanticipated faults and, in particular, multiple faults can be dealt with. This is based on the fact that the basic algorithm requires only the specification of models of correct behavior and, hence, makes no assumption about particular faults.
- New abnormalities, i.e. any observation that manifests a deviation from normal behavior, even if encountered never before, will be detected as a contradiction and, hence, contribute to the generation of solution candidates.
- The diagnosis procedure guarantees completeness and correctness of the results with respect to the phenomena captured by the models, the faults modeled, if any, and the observations provided.
- There is a formal theory that characterizes the correct and complete output of these diagnosis systems ([2]). Therefore, it is possible to provide provably correct diagnosis software for safety critical systems, in particular in the area of embedded systems where this requirement is often raised. Even if that level of software quality is not desired, the existence of the theory and a simple test algorithm (Quine's algorithm for computing prime implicates from 1955) allows for automatic testing of diagnosis output based on test cases.

- Model-based systems are generated, not programmed. As pointed out before, the only device-specific element is captured by the representation of the device structure. Often, it will be possible to import it from external, already existing sources, e.g. CAD systems. From this information and the elements of the domain-specific model library, a device behavior model can be derived automatically. The general diagnosis algorithm operating on this device model forms a diagnosis system tailored to the particular device. This means it is constructed without having to write a single line of code.
- Since the diagnosis framework can be bought from the shelf, the entire effort of programming and knowledge-base construction is in the establishment and maintenance of the model library. This is essential for solving the variant problem. From a practical and economic point of view this is feasible.
- Even if the actual runtime system is not a model-based one, model-based diagnosis can support programming diagnostics, for instance, for validation or as a starting point for the programmer who wants to violate the completeness of the model-based results in a controlled way. Also, the model-based technology can be used to generate the input to traditional types of diagnosis tools, e.g. by generating decision trees [3].

4.3 Software Platform for Model-based Solutions

In IDD diagnosis is only one of several addressed tasks, all of them occurring within the same product's life cycle. The initial tasks will be diagnosability analysis, FMEA, and on-board diagnosis. Given that these tasks can be solved individually by appropriate state of the art systems (e.g. [4], [5]), the challenge lies in providing

- a common software platform with components that are re-usable in different contexts, and
- the harmonization of models used for different tasks

The latter is ideally to be achieved by automated transformation routines. In particular the automated transfer of traditional quantitative models (used e.g. for simulation and control design) to qualitative models allowing for automated FMEA and fast, i.e. real-time, on-board diagnosis, is a central target. If indeed successful the re-use of existing model fragments for *different* tasks will reduce life cycle costs by a significant amount.

4.3.1 Architectural Requirements

IDD envisions three types of application settings. First, there is an integrated toolbox with its own graphical user interface and storage of models. Second, there are a variety of possible plug-ins to industry-adopted existing tools, e.g. MatLab, StateMate, ... Models will possibly be stored with these tools and the graphical user interface will be limited, if existent at all. Third, there is the (on-board) processing scenario for dedicated applications such as diagnosis and monitoring.

The Toolbox

The toolbox provides an interactive development environment for qualitative modeling and task-related testing of models. Its resides on a repository of model fragments that a modeler will use for creating new designs or elaborating existing ones. A component-oriented ontology is chosen to best address systems in the automotive domain. An open interface for the import of model fragments from other tools is a key requirement for the horizontal integration of various life cycle processes. Model fragments are assembled into system descriptions describing concrete variants of devices. Such system descriptions are suitable for task-related testing and for export to create dedicated on-board and other applications.

The Plug-Ins

An essential feature of the envisioned plug-ins besides their primary intended purpose is the use of data in different technical, but also different conceptual formats. The respective tool's native data store will be the source of models. Typically, these models will be quantitative ones in contrast to the qualitative ones required by the diagnosis-, prediction-, FMEA- etc engines. Hence, an automated transformation will be required before the main task can be run.

Dedicated (on-board) Applications

Dedicated applications are best characterized by the fact that they are intended for a particular variant of a device. A diagnosis- and monitoring application on an ECU is a typical example. Real-time- and memory requirements are of central concern here. System descriptions and real-time observations are the inputs and actions to be taken constitute the output.
Also diagnosis in the service bay fits under this umbrella. The only extension needed is that the diagnostic service equipment stores all relevant system descriptions.

4.3.2 Technical Requirements

While substantially different when viewed from the outside the three application settings share a substantial overlap of technical requirements that are used to define a core of functionality.
Fast consistency checking suitable even for on-board processing and handling of time as characterized by state of the art model-based diagnosis are the main characteristics of the processing at diagnosis and monitoring run time. Furthermore, a compact representation of models is needed. This is especially relevant for qualitative models that have to explicitly enumerate tuples of quite large relations describing behavior of components.
During the model building phase special emphasis is put on *automated* model transformation. Moreover, persistent data storage, version control of model fragments, and other, familiar requirements for model or software building platforms come into view. These are, however, not of central concern of the IDD project.

4.3.3 Implementation Principles

The IDD toolbox and plug-ins will be running on Microsoft Windows. Therefore, COM (component object model) was chosen as protocol for the interaction of (binary) components. It is our first cornerstone. All the engines, transformers, etc are implemented obeying this standard. This allows for the re-use of functionalities in different contexts, and in particular the three different application settings. The second cornerstone is given by the use of XML (extended markup language) for describing data in a uniform and exchangeable way. Many of our software components take XML documents as input and produce such documents as output.

For example, three important XML data interfaces are the description of quantitative information about systems extracted from tools such as MatLab, the description of qualitative information, e.g. after automated transformation, and the final system descriptions suitable for processing by diagnosis, FMEA, etc. COM and XML allow us to build task-related applications that are constructed from components which themselves are aggregated from even more basic components. The components in the layer directly under the application level we call Engines, our third cornerstone. So, there are (re-usable COM) components that encapsulate a diagnosis engine, an FMEA engine, a predictive engine, a transformation engine, etc. An important consequence of the choice of COM, XML, and Engines is that the resulting architecture is an open one, open at any desired degree down to the level of individual methods of low level objects.

4.3.4 Some Essential Components

ATMS (Assumption Truth Maintenance System)

The ATMS forms the inner core and implements fast consistency checking and handling of time. It not only tells us when an inconsistency is detected, but also which assumptions about system components working in particular modes at a time were violated. While still adhering to the basic framework of assumption-based truth maintenance [6], the employed technology has changed substantially making possible the implementation of on-board systems meeting real-time requirements [7].

Prediction

The predictive engine uses mathematical relations - called constraints – to derive values for system variables given observations or measurements about other system variables. It limits the effort by specifying appropriate foci of attention.

Model Compiler

The model compiler produces system descriptions (XML documents) suitable for processing by various engines. It assembles such system descriptions from descriptions of component types and the specification of structure in a device. The most space consuming part of such descriptions are qualitative relations which may contain thousands or even millions of tuples. A data structure similar to ordered binary decision diagrams (OBDD), but also suitable for direct constraint processing is used as a compact representation [8].

Diagnosis Engine

The diagnosis engine accepts a system description and a continuous stream of observations (measurements) as input and produces an assessment of the current situation by listing the best candidates for diagnosis, i.e. sets of pairs of components and their respective mode. It uses ATMS, predictive engine and a candidate generator capable of handling large search spaces which employs again OBDD type data structures.

FMEA Engine

The FMEA engine can operate in at least two different modes. First, it is able to enumerate effects of chosen sets of components faults. This allows one to fill tables which, for example, show the effects of all conceivable double faults in a device. Second, starting from given symptoms the FMEA engine can compute a complete set of possible combinations of causes that could produce the symptoms. This mode is very similar to the operations of the diagnosis engine, and indeed these two engines share the predictive engine and the candidate generator.

Model Transformation Engine

As already pointed out automated model transformation is required to obtain qualitative models. Behavioral and structural descriptions are extracted from tools such as MatLab/ Simulink, converted to quantitative XML form and then transformed into qualitative descriptions through a process called task-dependent model-abstraction [9].

4.3.5 Current Status

On the component level the IDD consortium has chosen OCC'M's Raz'r [10] as a basis for implementation. It provides state of the art model-based systems software packaged into COM-components and supplied with XML-interfaces. This allows for further extensions as needed by the consortium requirements. Currently, all of the above mentioned components with the exception of the model transformation engine are supplied by OCC'M. The model transformation engine is central and touches on still open research questions. Therefore it is a main subject of the consortium's current activities. While the structural extraction poses no deep problems, the behavior part is more difficult. Nevertheless, the essential theoretical work has been done, and we expect to finish as scheduled.

On the tool level the specification of tools for integration into the new design process is in the stage of use case specification. Several tasks in the new design process have been identified where computer support through model-based solutions will make a big difference. The implementation of these tools will be achieved by mapping to functionalities at the component level.

5. Guiding Applications

IDD will use a number of guiding applications with the goal to demonstrate how the diagnostic tasks described can be performed by using the new process and the new tools architecture. Furthermore, we aim to demonstrate how additional advantages of the new method can be achieved, e.g. optimization of sensor placement or deeper diagnostic performance. Thereby, the guiding applications serve, on the one hand, as case studies for the application of the new techniques and, on the other hand, as test cases and demonstrators of the results of the project.

The guiding applications chosen cover on the one hand different mechatronic systems with central ECU-functions, and on the other hand the general application of diagnostic tasks to multiplexed architecture systems.

The choices have been made with respect to significant complexity, relevance to diagnostics, representativeness of typical design process and manifestation of advantages of the new design process and diagnosability analysis.

5.1 Mechatronic systems applications

The Air delivery system and the Common Rail Injection System are the guiding applications chosen by Fiat CRF and Magneti-Marelli.

The first application regards the air delivery system for diesel engines, comprising the exhaust gas turbocharging system and the exhaust gas recirculation system (EGR).

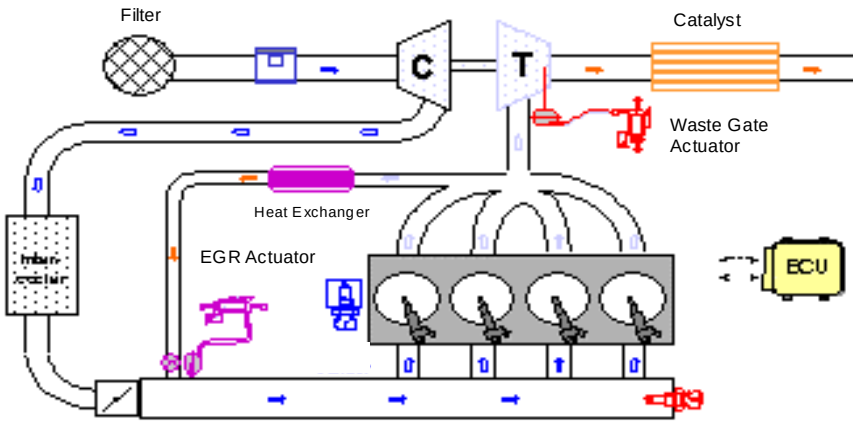

Fig. 5-1: Guiding application: Air delivery system

In exhaust gas turbocharging, some of the exhaust gas energy is used to drive a turbine and by this means a compressor which draws in the combustion air, compress it, and then supplies it to the engine in order to increase the engine efficiency. Furthermore the system includes an intercooler, described below in the context of the cooling system.

The exhaust gas recirculation (EGR) is used in Diesel engines for the purpose of reducing emissions of NOx. The fraction of exhaust gas which has already combusted is recycled through a control valve from exhaust to the engine intake system in order to reduce the peak combustion temperature. Since the formation of nitrous-oxides increases proportionally with the combustion temperature, EGR as a temperature-reducing measure is a very effective method of reducing NOx.

As a further application the common rail injection system will be used. Purpose of the Common Rail Injection system is to provide injection of a well defined amount of fuel spray as a function of time (directly) into the combustion chamber of a diesel engine. To achieve this, injection pressure is generated independently of the engine's rotational speed by storing pressurised fuel in the rail.

The Cooling system is the guiding application chosen by DaimlerChrysler AG.

Cooling systems play a central role for the thermodynamic processes inside a combustion engine. The cylinder pipe, cylinder head, valves and piston, which surround the hot combustion chamber, have to be cooled intensively. To keep the serviceability, about 30% of the burning heat should be discharged.

The cooling system regarded here furthermore includes an intercooler, which on the one hand raises the efficiency of the engine, by cooling the compressed air and hence increasing the air charge rate, and on the other hand decreases NOx emissions by keeping the combustion at lower temperature.

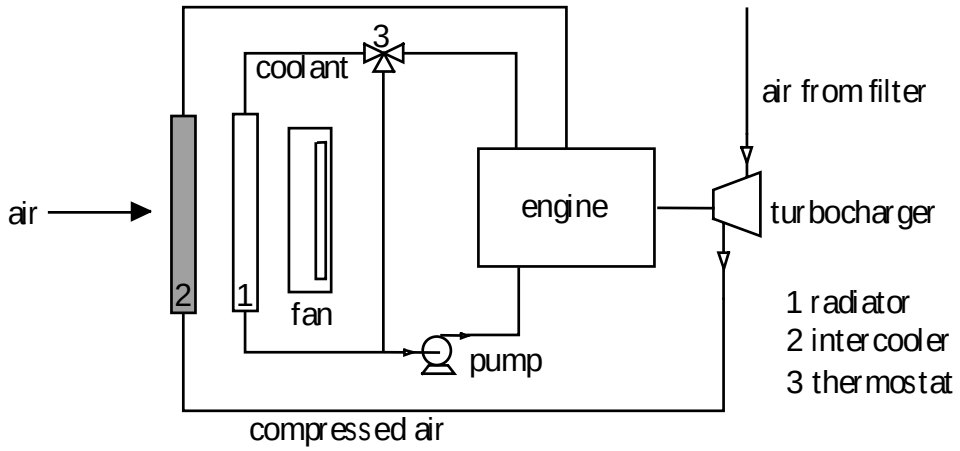

Fig. 5-2: Guiding application: Cooling system

The Air conditioning system is the guiding application chosen by Peugeot Citroën PSA.

Air conditioning is aimed to perform two functions: to ensure passengers' climate comfort by air processing and to ensure visibility through the glazing of the passenger compartment. Materials used in the car, extension of climate variations and response time require an important installed power. Dysfunction of sensors or actuators does not presently allow a climate comfort optimization in case of defect.

Air conditioning system consists of two loops that supply a cold heat exchanger and a hot heat exchanger; air processing is performed in the equipment that supplies processed air to the passenger compartment.

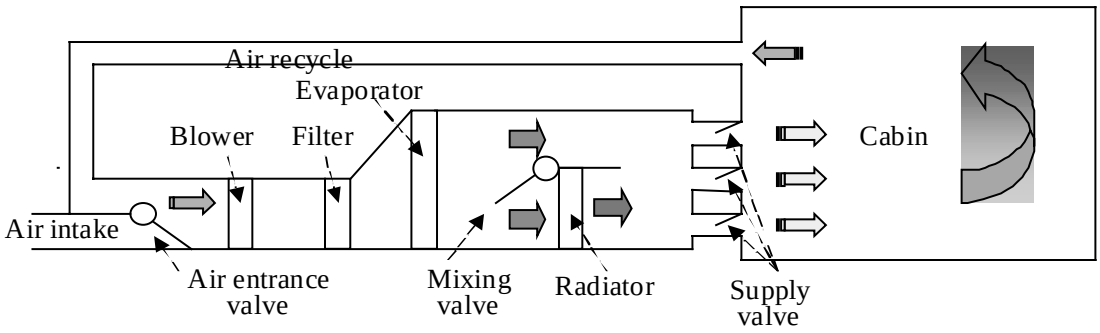

Fig. 5-3: Guiding application: Air conditioning system

According to the model-based approach, the above mentioned steps (system decomposition, generation of models library and composition to system models) are necessary and are currently a focus of work in the project. The next steps are to complete the definition of design scenarios for demonstration and to perform their illustration by the new tools architecture.

5.2 Multiplexed architecture application

The multiplexed architecture is the guiding application chosen by Renault.

The application concerns multiplexed embedded systems. The objective is to demonstrate the ability of detecting a growth of diagnosis complexity due to some architecture modification during the design phase. In order to be able to evaluate the impact of an operational architecture modification over the number of causes for one client effect, it is intended to connect model based diagnostic tools and architecture description tools. The design engineer can run a program directly from the new architecture description and get quickly a synthesis of what function is impacted in terms of diagnostic complexity.

Example

Imagine a simple car in production with no known electronic problems. The car has an electronic motor control and gearbox. The gearbox reads the real torque output by the motor, and uses it to select when to perform a requested shift. When the motor control operates out of a specified range of parameters, it outputs an *invalid* signal for the torque, and the gearbox ignores shift requests until a valid torque is seen. A new car is being designed that will use this drivetrain with an ESP unit from a more sophisticated car design. The ESP regulates torque, shifts and breaking in turning conditions to optimize adhesion. The ESP also receives the torque output from the motor. The designer views the system at a high level and is unaware of these details. A situation can now exist where, for a minor problem, the motor control stops outputting torque. In this condition the ESP abruptly stops functioning with possibly disastrous consequences. When the designer grafts the ESP onto the powertrain units to make a new design he runs a diagnostic check for fault analysis. The analysis should identify the possible fault chain leading to an invalid torque signal and pursue it to the minor motor control errors that can cause it. The designer, now alerted, identifies the need to change the control motor data processing to remove the hazard.

Elements and Interfaces

The components represented on the following picture are ECUs, busses, functions (EF = elementary functions) and data. Frames can be represented in the description of the bus and ECU terminals. The objects and their behaviours are determined by the means of a relational database, which is part of the standard development process in Renault for multiplexed architecture. It is called OASIS. The objects are fetched in this database so that the structural interface poses no problems.

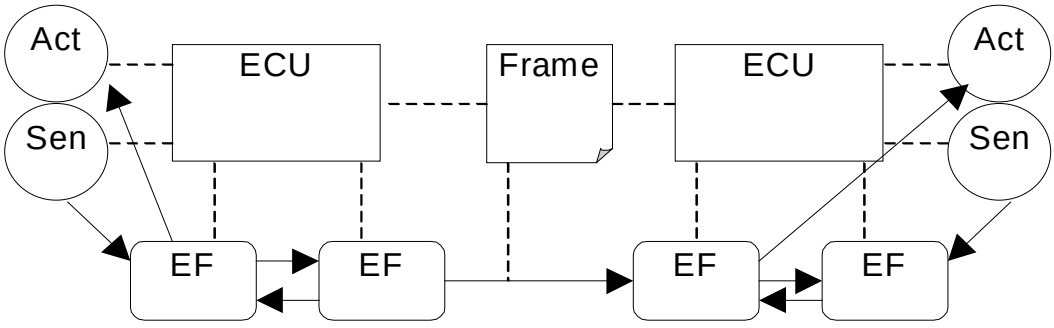

Fig. 5-4: OASIS elements and Interfaces

6. Conclusions

For a better integration of diagnostic tasks in the surrounding design process, which is the goal of the IDD project, a frame for a new design process and the requirements for a new tools architecture have been presented. By this means the designer will be supported for performing in an interleaved way different activities, like design of the physical system, design of control algorithms, support for FMEA, analysis for diagnosability and derivation of onboard diagnostic software.

The next steps in the project are to complete the tools architecture based on the requirements, to complete the guiding applications in order to prove the advantages of the new process and architecture and finally to give contributions to European standards for the automotive design process.

7. References

[1]: J. Bortolazzi, St. Steinhauer, Th. Weber: Development and Quality Management of In-Vehicle Software, 9. VDI – Berichte 1547, „Electronic Systems for Vehicles“, 2000

[2]: J. de Kleer, A. Mackworth und R. Reiter: Characterizing Diagnoses and Systems. Artificial Intelligence, 56, 1992
and:
O. Dressler und P. Struss: The Consistency-based Approach to Automated Diagnosis of Devices. In: Brewka, G. (ed.), Principles of Knowledge Representation, CSLI Publications, Stanford, pp. 267-311, 1996.

[3]: F. Cascio, L. Console, M. Guagliumi, M. Osella, A. Panati, S. Sottano, D. Theseider-Dupré: Strategies for on-board diagnostics of dynamic automotive systems using qualitative models, AI Communications, June 1999.

[4]: P. Bidian, M. Tatar, F. Cascio, D. Theseider-Dupré, M. Sachenbacher, R. Weber, C. Carlén: Powertrain Diagnostics: A Model-Based Approach, Proceedings of ERA Technology Vehicle Electronic, Systems Conference '99, Coventry, UK, 1999

[5]: C. Price: Function-directed Electrical Design Analysis, AI in Engineering 12(4), pp. 445-456, 1998.

[6]: J. de Kleer: An assumption-based truth maintenance system, Artificial Intelligence 28, 1986

[7]: M. Sachenbacher, P. Struss, R. Weber: Advances in Design and Implementation of OBD Functions for Diesel Injection Systems based on a Qualitative Approach to Diagnosis, SAE 2000 World Congress, Detroit, USA, 2000.

[8]: R. Bryant: Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams ACM Computing Surveys, Vol. 24, No. September 1992

[9]: M. Sachenbacher, P. Struss: AQUA: A Framework for Automated Qualitative Abstraction. In: Working Papers of the 15th International Workshop on Qualitative Reasoning (QR-01), San Antonio, USA, 2001

[10]: Raz'r Version 1.6, Occ'm Software GmbH, see http://www.occm.de/