Hierarchical and Distributed Diagnostics: Fault Tolerance from a System Perspective[1]

Annamária R. Várkonyi-Kóczy, Gyula Román, Tadeusz P. Dobrowiecki

Department of Measurement and Instrument Engineering
Technical University of Budapest
H-1521 Budapest, Műegyetem rkp. 9
E-mail: {koczy,roman,dobrowiecki}@mmt.bme.hu

1. Introduction

In applications the burden of responsibility is increasingly taken away from the user and passed onto the system. Such demand requires highly dependable equipment and subsystems of well defined flexible functionality. Technology, and design techniques cannot reduce further the likelihood of failure. Costs of required reduction are usually prohibitive, so some other means are needed to upgrade the system's dependability. Fault tolerance, fault tolerant design offer a suitable framework and particular working solutions.

In the field of fault tolerant design and computing in last decades most efforts were concentrated on the fault tolerant digital design, and powerful techniques, such as correction codes, self-checking logic, and module replication have been developed. These techniques attempt to ensure a continuous error-free operation, using redundancy artificially integrated into the system. In numerous applications the demand of economy does not make it possible to ensure continuous totally error-free operation. Breakdown and some amount of downtime for repair is unavoidable. This draws attention to fast diagnostics, reconfiguration and repair methods.

With the spread of computers and the revolutionary development of network facilities highly distributed systems evolved. Such systems permit to reconsider the idea of the fault tolerance at a higher level of abstraction. Distributed architecture directly leads to a distributed diagnostics and generally speaking there is much more potential in it than what is purposefully utilised. In recent applications systems are very complex, typically containing multi-functional components. These components directly realise functional redundancy, which should be considered for the improvement of a fault tolerant behaviour. In addition, the role of computing elements drastically increased, and such systems offer extra computing capacities to be exploited for diagnostic purposes.

To represent formally the functionality of a system, a suitable modelling framework, describing its parts and the logical relationship between them, is needed. This problem was addressed by some researchers who offered functional maintenance as a possible solution. Though functional maintenance deals with the functional roles of the components, it is not suitable for large hierarchical system modelling. Present paper addresses the issue of system modelling at functional level. It seeks means to

[1]This work was supported by the Hungarian Scientific Grants Nr. F-007414 and T-014403.

express the functional decomposition of the system into a suitable hierarchy, using primary concepts of 'services' and 'requirements'.

Authors present an efficient modelling scheme to describe the decomposition of the overall functionality of a sophisticated system and propose a distributed diagnostic architecture. Such architecture permits a better utilisation of the inherent potentials and distributed nature of knowledge and resources to solve efficiently the reconfiguration and repair problem, to reduce the Mean Time To Repair (MTTR), and to offer fault tolerance at lower costs. Furthermore, it makes easier to handle complex systems containing entirely different parts e.g. computer systems, networks, industrial technologies uniformly. The dynamic reconfiguration algorithm is demonstrated through a simple example.

2. Designing Systems Which Fail

Designed system always fail, earlier or later. It is not necessarily a fault of so called system designer. A sophisticated system represents functional and physical hierarchy of such depth that specifying and designing it at a higher level necessarily assumes ready-to-use out-of-shelf modules of lesser complexity to build with, let it be hardware or software. These modules will ultimately fail bringing down the whole design. In vain designers will wash their hands pointing to a faultless, even formally verified design, the truth is there is no functioning system in the hands of the user. The basic question of system design 'how to design a system to comply with a given specification ?' turned under the pressure of the reality into 'how to design a system which will fail to comply with a given specification ?'. But first of all we should investigate what does it mean that a system fails.

Failure of the system means generally that the system supports no more its specification, used to design system functions. That such situation occurs is directly observable in the system behaviour with respect to its external environment (e.g. also to the user). These observables are symptoms of faults within the system, and if the system is sophisticated and hierarchically built, symptoms bring only indirect, partial and uncertain knowledge about the nature of the problem. Symptoms rely the information that at least some of the functions of the system changed, due to the changes in the system structure, somewhere deep in modules assumed faultless in design.

3. System Modelling and Fault Diagnosis

Failure of systems is related to a number of well defined engineering activities. The task of watching system's behaviour and finding out that there is something wrong with the system's behaviour is that of 'system monitoring'; processing symptoms until a finger can be put on the faulty module is 'fault diagnosis'; substituting the wrongly functioning part of the system to resume its normal function is 'repair'. Although the success of the diagnosis depends ultimately upon the particular knowledge about the 'good' and 'bad' behaviour of the system under study, diagnosis in general is a richly structured activity with typical reasoning steps and strategies.

Basically a diagnosis runs through the sequence of

- 'symptom detection' generating symptoms from observable quantities;

- 'hypothesis generation' (more hypotheses seem feasible due to uncertain observations);
- 'hypothesis discrimination' pruning candidates, i.e. leaving only those hypotheses closely related to the case at hand, usually by some kind of testing injecting new knowledge into the process, finally
- 'generating explanation' when no more information can resolve the problem better.

Although dealing with faults reveals a common structure at the very high level of abstraction, concrete methods and the advanced character of diagnosis depend upon concrete systems and the formalism of their models.

System models differ primarily in level [16]. Shallow, skill-based level leads to no more but an almost routine performance. Such model is usually encoded as rules-of-thumb, direct patterns, mapping directly from observations to hypotheses. A more conscious diagnosing behaviour can be produced by a rule-based model, usually a set of rules (fuzzy or other natural language models) with included inference mechanism to be run in the course of diagnosis. Knowledge-based level yields a deep model and a model-based diagnosis, needed in cases when common patterns in stored knowledge form do not exist and reasoning should start from the so called first principles.

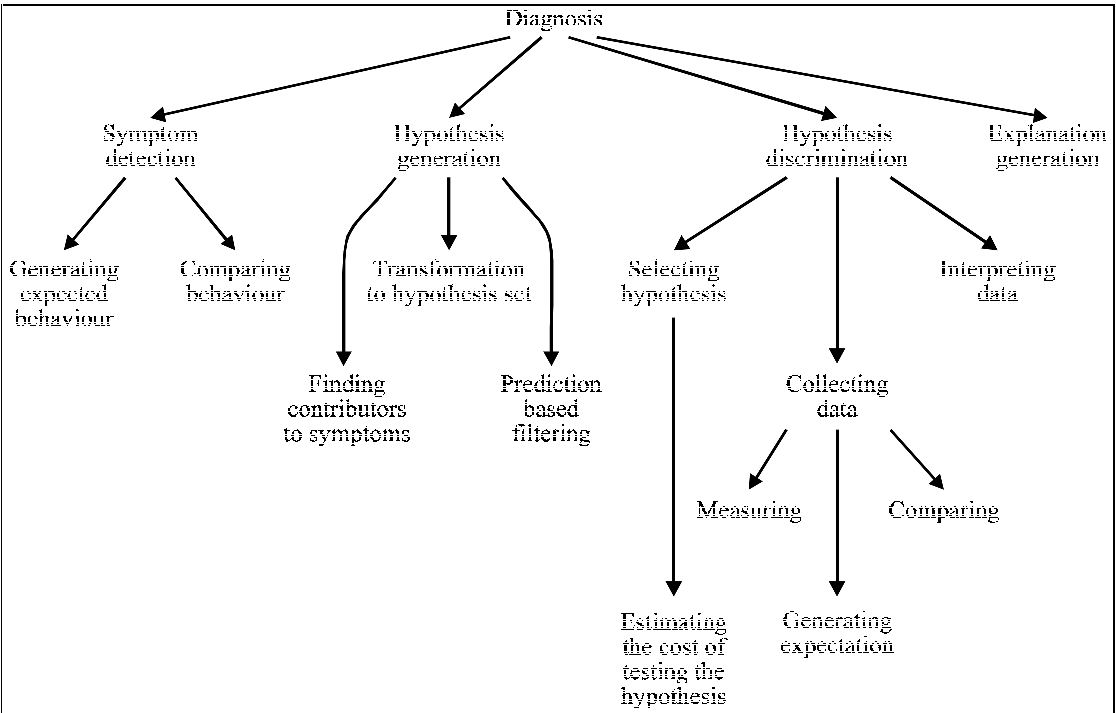

Fig. 1. A more detailed decomposition of the diagnosis task [2]

In a number of engineering problems fault diagnosis and isolation techniques generally rely on a precise mathematical model (i.e. a deep model) of the process and on pr enumerated symptom-fault patterns known as fault signatures [3,13]. The estimation methods used for symptom processing do not accommodate easily other more shallow system models.

More levels are possible if the system under study lacks an exact and still manageable description. This is the case of sophisticated large-scale industrial processes, biological systems or systems commonly found in computer science (especially in Artificial Intelligence related problems). Deep system models are qualitative in nature and based in such applications on a hierarchy of structure and behaviour.

Qualitative models derived from various approaches in qualitative physics are well suited for diagnosis, due to the fact that they are inherently constraint-based, and system failures, faults, symptoms, etc. are easy to be interpreted in terms of unfulfilled constraints and differences in qualitative values of the state variables [10,11,12,15]. An interesting approach to the qualitative model-based diagnosis is an automatic rebuilding of qualitative models when the system fails and faults are identified [14]. Diagnosed faults are in such case injected into the model, which until repair predicts correctly further evolution of the system.

Using a single modelling approach is usually not enough. Uncertain missing knowledge about complex phenomena makes any kind of engineering task almost hopeless. Difficulties in modelling call in such sophisticated cases for multiple perspective and for multiple models of various formalism and level. Integrating multiple models is not easy [17]. The user must understand well and represent this knowledge in a usable form how to decide when to shift between models, how to identify where to shift to (level, formalism), how to translate between models, and how to integrate information from different domains.

A separate challenge is a real-time fault diagnosis, considering that the majority of formal qualitative methods is exponentially difficult, and the only possibility to speed-up the process lies in using efficient heuristics. Consequently only soft and slow real-time problems had been tackled, successfully where the value of the diagnosis is degrading as time passes on. An interesting approach is to tackle varying deadline problem with the multiple models, using the fact that models at different levels and of different formalism possess different run-time requirements, and that way a good trade-off between the accuracy of the diagnosis and keeping the real-time deadlines can be achieved.

Coupling diagnosis with the subsequent repair process is still an open question, due mainly to the different character of the interaction between the diagnosis system and the system under study in both phases.

4. Fault Diagnosis and Fault-Tolerance

Let us reiterate to the initial question of how to design system which will fail. System with a built-in diagnostic component and repair facility will be immune to failures, at least to the extent of its diagnostic and repairing abilities. Self-diagnostics systems are already abundant and in certain fields almost mandatory, on the other hand self-repairing systems are still rarity. There is however a way in-between - fault-tolerant systems - increasing the flexibility of the system's usage but still not accepting costs and trade-offs of the fully self-repairing design [6,8,21,22].

Fault tolerance is an ability to continue executing tasks despite the occurrence of failures. Of course system will be resistant to faults formally specified and embedded into the system design. System equipped with resources strictly matching the requirements of its specification cannot achieve anything. Only increasing the system resources, introducing redundancy into the design permits the usage of some particular fault tolerance strategy [8].

Redundancy - a basic requirement of the fault tolerance - can be implemented in many different ways. The most simple is hardware redundancy, where system modules are multiplied. Computer systems can base also on software redundancy and generally on (run)time redundancy. Working at a higher level of abstraction the fault tolerance of any kind of system can be achieved with functional redundancy. Analytic redundancy is also mentioned in fault diagnosis of industrial systems.

The primary problem with the fault tolerant system design is that despite a clear and system independent notion of system reliability, dependability, etc. particular techniques and level of fault tolerance achieved will depend heavily upon the character of the system and model(s) used to describe its behaviour.

4. Fault-Tolerant Systems - Basic Notions [8]

A system is dependable when its services can be trusted. For a system to be dependable, it must be available (e.g. ready for use when we need it), reliable (e.g. able to provide continuity of service while we are using it), safe (e.g. does not have a catastrophic consequences on the environment), and secure (e.g. able to preserve confidentiality). At design time the dependability can be increased through fault avoidance techniques. At implementation time the same aim can be achieved through fault removal techniques. At execution time, on the other hand, fault tolerance and fault evasion techniques are required.

A system is said to have a failure if the service it delivers to the user deviated from those prescribed for a specified period of time. A fault is the failure of a component of the system, a subsystem of the system, or another system which has interacted or is interacting with the considered system. Every fault is a failure from some point of view. A fault can lead to other faults, or to a failure, or neither. The observable effect of a fault at the system boundary is called a symptom.

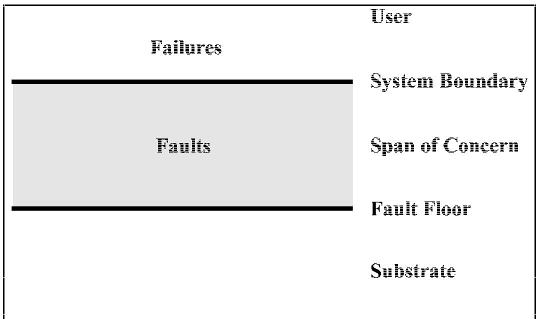

Fig. 2.: Failure regions [8]

From the user's point of view two system boundaries are of interest: the system boundary, and the fault floor (see Fig. 2.). The first is the borderline above which the user expects failureless operation; and the second refers to the borderline, below which the faults are no longer of interest for us. Faults below the fault floor are indistinguishable, either because they are not fully understood, or because they are too numerous. Atomic components lie at the fault floor. Faults emerging from atomic components, but not faults within these components are of interest for us. Failures occur when faults reach the system boundary. Practically the span of interest is the area, in which faults are considered.

It is impossible to make a system that tolerates all type of faults, so it is vital to consider the faults throughout the requirement definition, and during system design. To make the numerous faults practically handable, it is essential to group the into meaningful categories. These categories can be based on locality, effect, and cause. Some of the categories are listed below: operator faults, replication faults, malicious faults, external faults, environmental faults, user faults, etc.

We have to consider two important fault attributes: observability and propagation. The first deals with the effects of the fault, i.e. if the effects can be recognised as different from normal behaviour, we say that the fault is detected, else it is latent. The second attribute addresses the propagation of faults. If the fault propagates, it is said to be active, else is dormant. Fig. 3. shows the relationship between detected, latent, dormant, and active faults.

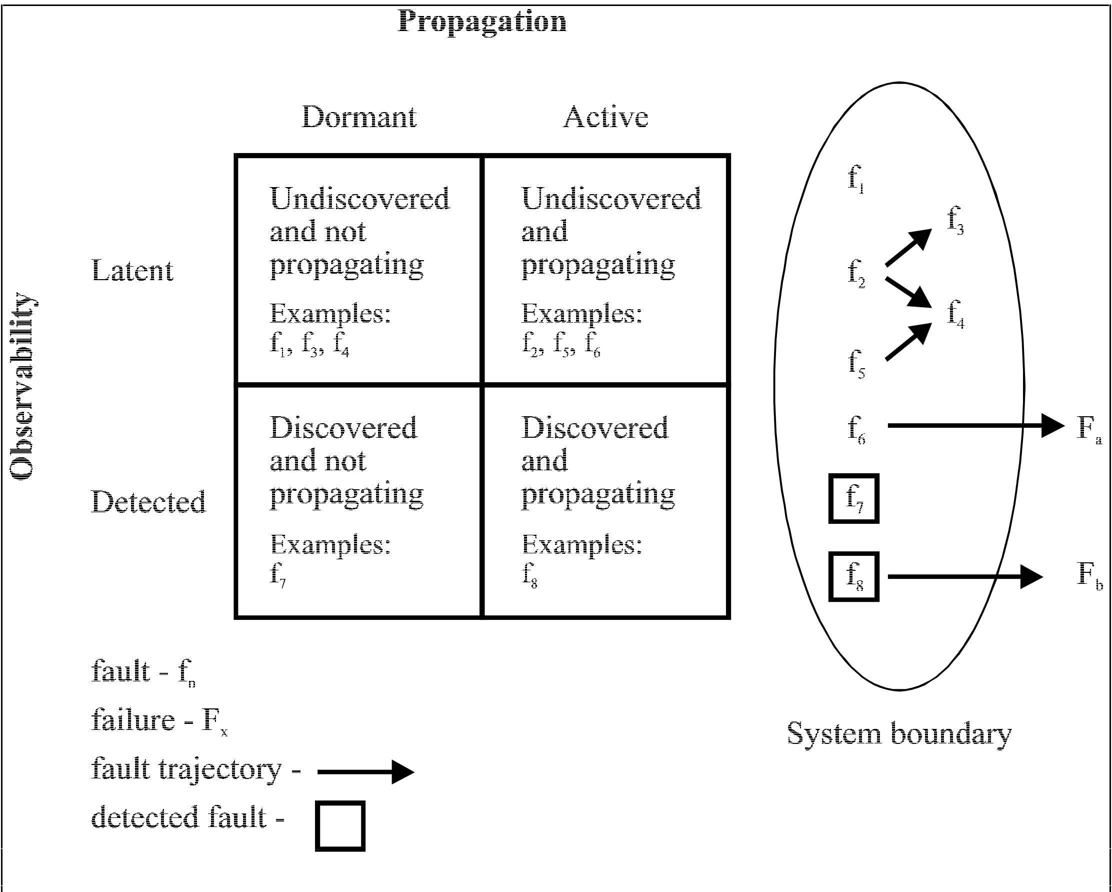

Fig. 3.: Attributes of faults [8]

5. Fault-Tolerant Systems and the System Model

When fault tolerant systems are mentioned the first picture that flashes into our mind is that of a digital system. The very nature of such systems allows for fault tolerant methods, and this is the field that has inspired the most research in fault tolerant techniques.

Why digital computer system dominate the fault tolerant techniques? Because they have special characteristics that determine how these systems fail, and what fault tolerant methods are the most appropriate. In addition, the conditions can be well kept in hand, which is very important when dealing with large systems.

The most important characteristics that digital systems hold are:

1. they operate in discontinuous steps,
2. they encode information,
3. they can modify their behaviour based on the information they process.

The second property has strong implications, i.e. digital systems can store information without change for a long period of time. In contrast, in analog representations values are subject to value drifts. In addition, it also allows for redundancy, several codes can represent the same value [8].

Fault tolerance techniques are also applied to the distributed computer systems. Issues which must be solve in designing fault tolerant open distributed systems, concurrency control and redundancy management in particular are described in [1,7,19,20]. Fault tolerance of distributed real-time systems, assuming solely fail-silent nodes and omission failures in communication channels can be achieved via special protocols [9].

Fault tolerant behaviour can be invoked in systems modelled with qualitative models. Symptoms produced by the sensors monitoring the process are used by the model-based reasoner which diagnoses faults and plans repairs. Repair can be realised with a control type strategy - adjusting various process parameters - and can handle faults such as deteriorating components. Functional redundancy strategy explores potential functions of existing components in a slightly different way from the original design. However such functional redundancy must be added to the system with a functional redundancy designer [10].

An interesting is the idea of analytical redundancy instead of hardware redundancy in industrial systems with sensory measurements. Measurements are processed analytically to compute the value of the desired variable. Mathematical model is used as a duplicate of the physical system for monitoring and comparison purposes [3].

Summarising the situation in fault tolerant design from a system model perspective we can observe that digital systems, due to their advantageous properties can be equipped with a full spectrum of fault tolerant solutions. These solutions however draw from low level redundancy, functional redundancy is not explored and the distributed and real-time systems pose already a problem. In non-digital systems fault tolerant techniques are scarce and usually the nature of the problem does not permit to draw a too strict analogy with digital system solutions. Generally despite the inherent hierarchical concept of faults and failures fault tolerant techniques are applied to homogenous single-level systems.

6. Complex systems in measurement and modelling

From the point of view of system modelling and thus development of fault tolerant systems complex industrial-like systems mean real problem. These systems are directly coupled to the physical process, which they observe running tasks of system monitoring, diagnosing, advising or even control. Dynamic character of the application requires real-time functioning regime, and the space-scale of the processes to be actively observed makes the designed system inherently distributed. Monitoring system runs usually round-the-clock, meaning that its resources are finite and have to be economised as well as possible.

The most important characteristics, however, is that the considered systems are inherently heterogeneous and hierarchically organised with well defined duties and restrictions at any level of the hierarchy. Hierarchy comes from the hierarchical model of the process and from the decomposition of the goals of the overall system [18]. Heterogeneous character is strongly related with the fact that information is flowing between an analogue and a digital environment, and that the diversity of goals results in similar diversity in the functions of the digital nodes of the system.

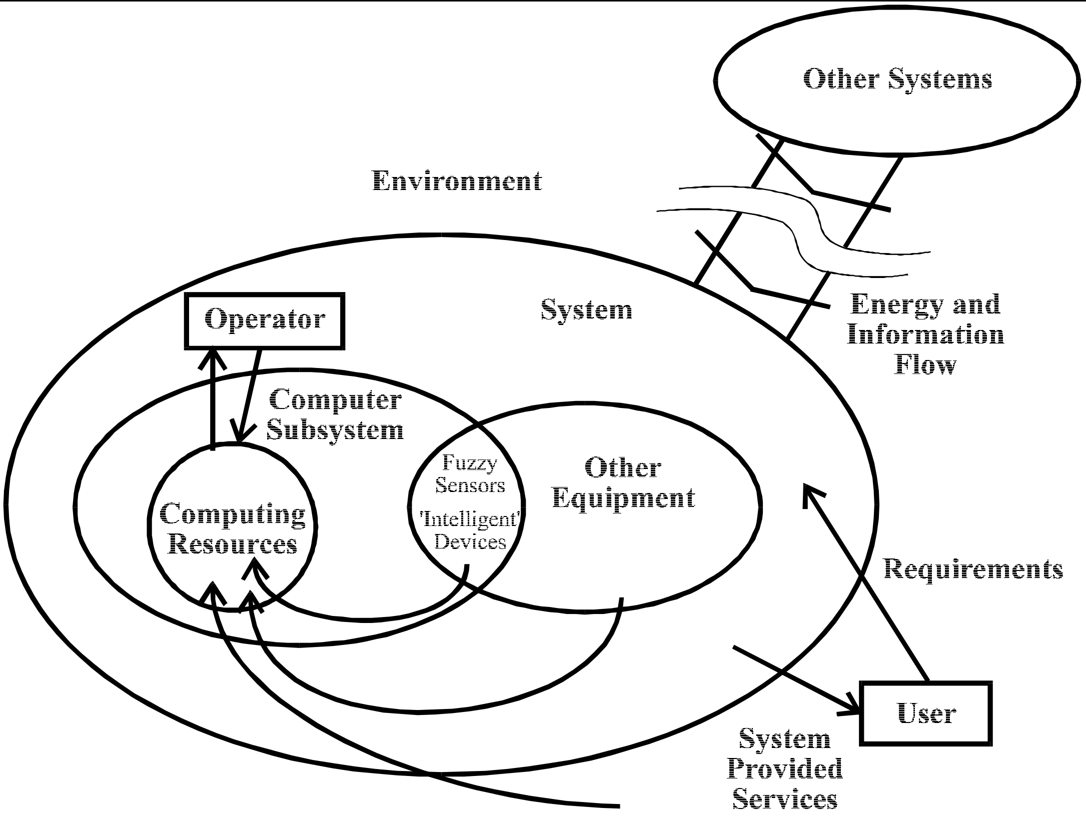

Fig. 4.: Spectrum of problems in industrial systems.

Fig. 4. grasps the essence of a complex industrial system. In this model the system is closed, and the communication with other systems is via the environment, which in turn, may distort the information and energy flow between the two systems.
When a user faces such a system, he/she formulates his/her wishes as requirements towards the system, and the system reacts to these requirements with services. From the figure it is apparent that a modern system in addition to traditional equipment, also contains computing elements (computer subsystem), and the role of this part is

gradually increasing. An interesting family of equipment has appeared on stage: the intelligent devices, (e.g. fuzzy controllers), which in addition to their traditional capabilities incorporate computing power.

In such systems it is very often the case that the system's resources are shared by numerous (often competing) users. Another feature of such systems is that they posses multi-functional elements, i.e. resources that can provide several services, some of them concurrently, but some of them exclusively. In these situations if care is taken in assigning the resources to users and tasks, the unnecessary performance degradation can be avoided.

If we analyse these features of recent distributed systems the following requirements can be formulated about fault tolerance:

- it must admit the description of system models at a generic, high level abstraction level;
- it has to cope with complexity problems both from model representation (memory) and model interpretation (algorithmic) aspects;
- it must support the extension of existing systems with reasonable efforts; and as one of the most important
- it should be able to use existing redundancy.

Although, classical fault tolerant methods proved to be enough sufficient in classical homogeneous systems, in recent complex, hierarchical inhomogeneous systems with differences in the modelling, depth, state, using different kind of redundancies, having differences in the requirements, timing, etc. they can offer reliable behaviour only for parts of the distributed system and not for the whole system (even in case of all parts can be stated as fault tolerant ones). So these methods fail or can be used with such a high burden of costs which can not be afforded neither by the designers nor by the users. All of these press designers and engineers to look for new methods, for non-traditional fault tolerant solutions. Adding redundancy in system intelligence means to combine artificial intelligence with fault tolerant methods. That means the use of the same notions with new meanings.

7. The basic problem

We consider a *distributed system* (or simply *system*) to be a set of *loosely coupled*, interacting *nodes*. Each node is a *resource* that performs given *services* to achieve the *global service* of the system. The *global service* is considered as satisfying a set of *interrelated requirements*. The notions *"requirement"*, *"service"* and *"resource"* are present in *any* distributed systems, so they can serve as a starting point, which is independent from the exact type and properties of the distributed system.

The system operates properly, if any requirement Qi is fulfilled by an appropriate service Si and the interrelations between requirements, services and resources are also satisfied. If Si initially satisfies Qi, and later Si becomes unable to satisfy Qi, Si is called a *fault*. This definition is a general *functional fault model*, which appears at a high abstraction level. A fault is *masked*, if its effects do not cause changes in the global service (or global behaviour) of the system. If a fault cannot be masked, it becomes a *failure*.

8. The definition of the model

To cope with algorithmic and memory complexity problems of representing and interpreting models of complex systems, the proposed modelling method uses hierarchical decomposition (see fig. 5.). The decomposition consists of *levels*. Each level represents an abstraction of the *whole* system. Any level is built of one or more *planes*. At each plane the model has a given structure built of *components* (defined in the following sections), and planes are connected to each other in a well defined way [4, 5].

Associated with each plane there is a diagnostic center, which observes the operation of the associated plane and maintains the services of the particular plane towards the upper levels. Simple components (which are not decomposed further) are observed directly and individually (e.g. via sensors in an industrial technology, by exception handlers or memory management units in a computer environment). In the case of compound components (or simply *compounds*, which have a decomposition at another plane) information is delivered by the diagnostic centers associated to the lower level plane holding the decomposition.

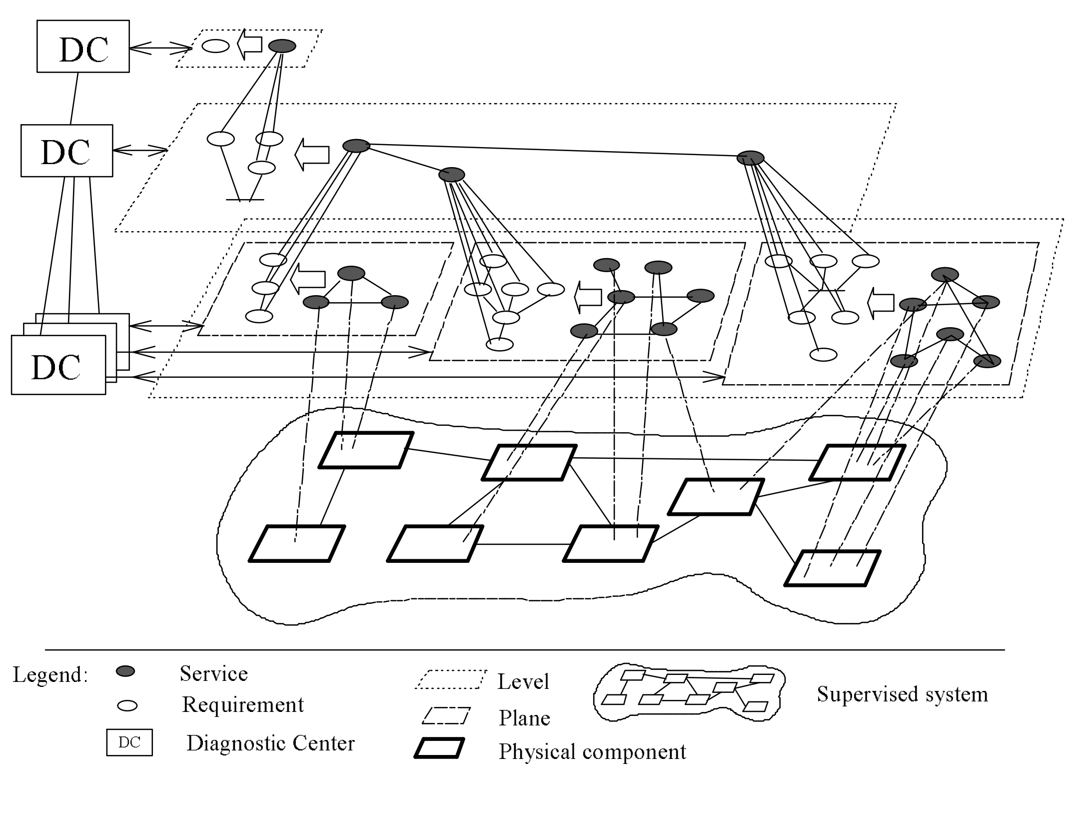

Fig. 5.: The structure of the system model

The diagnostic centers in a real situation can be *distinct physical computers* (which may be advantageous in distributed systems covering a large area) or can be *conceptual*, if diagnostic centers of more planes are placed to the same computer (which may be advantageous in smaller scale systems), though it makes no difference in the introduced concepts. The diagnostic centers are supposed to be *hard cores* [23], so they never get faulty. It means that they must run on reliable hardware architectures.

The hard core requirement can be significantly eased, if there are more physical diagnostic centers present, and they are designed to be able to partly or fully substitute each other.

At each plane the model contains the following elements:

- *Requirements* define the set of those tasks which the given plane must perform. Different requirements belonging to the same plane form a structure expressing that requirements (similarly to the steps of recipes for industrial technologies or in a cookbook) cannot be satisfied in an arbitrary order. The requirements at a given plane are ordered in a directed graph, whose nodes are requirements and whose arcs are predefined types of causal dependencies between requirement. Requirement structures are interpreted by a simple algorithm called IRS presented in [4, 5].
- *Service types* are the features, which can be used at a plane to achieve a set of requirements. A single service type may fulfil more requirements, even at the same time.
- *Resources* are the *components* of the supervised system, which are capable of providing different types of services. Resources can be *physical* (like a computer hardware) or *conceptual* (like a software module). Any resource has an associated set of service types, and any service type can be associated to more resources. Resources can be considered as real, existing parts of the supervised system, while requirements and service types are specifications and models for the operation of resources.

Properties of requirements, services and resources are described by associated sets of *attributes*. The basic method for maintaining fault-tolerant operation is called *attribute mapping*, where attributes of requirements are matched to attributes of available service types to find out which services can fulfil the given requirement and to instantiate new services by matching resource attributes to the given service type. In general, an attribute is described by a constant function or by a function of other attributes.

A new *service instance* (or simply *service*) *si=(Si, Ri)* can be generated, using a resource *Ri* and a *service type Si*, with exactly the same attributes as defined in *Si* and assigning values to its attributes by using the attribute value of *Ri*, which is a resource capable of providing services of the type *Si*.

When the requirement structure of a plane *pi* is satisfied, *pi* is said to provide its *global service*, since planes appear as a single service at a higher plane. To each *pi* there is an associated *diagnostic center* (DC), which maintains this *global service* of the plane, even if certain *requirements cannot be satisfied*. The DC fulfils this task by continuously monitoring the state (expressed by attribute values) of the services and resources at *Pi* and if certain attributes change, the DC has to maintain the global service of *Pi* by satisfying the requirement structure of *Pi* by generating new service instances instead of those, which fail to satisfy an element of the requirement structure.

Figure 6. demonstrates the attribute relation between requirements and services. The services are mapped onto requirements based upon the attributes: the diagnostic centers must find services that have attributes with the same name as the requirement to be satisfied. (The attribute names are unique, they represent a certain property of the

requirement and/or the service.) Based upon the possible of values that an attribute can take, they are classified into three categories:

- value type,
- enumeration type, and
- interval type.

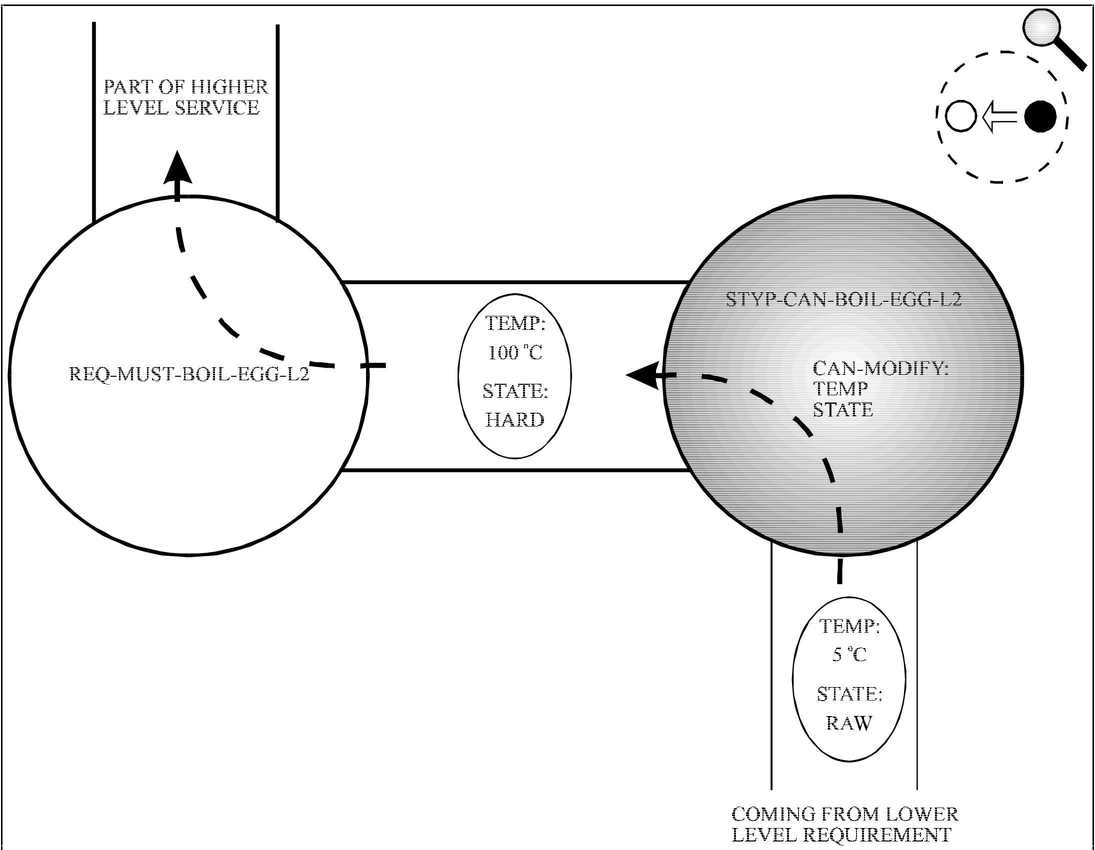

Fig. 6.: The attribute relation between requirements and services

When the diagnostic center has to decide whether the service can satisfy the given requirement, it utilises the so called attribute matching algorithm to get the answer. The algorithm tries to find services that satisfy the relations imposed on the matching. Several relations can be prescribed: we can specify exact, or approximate match. In the case of exact match, the system must ensure that the attribute values of the requirement and the service are identical, while in the other case the DC is satisfied with close values (though the quality of the service strongly depends on the 'closeness' of the values).

It is crucial that the values of attributes can only be changed by services, and these possible changes are characterised in the service-description itself. The description contains which attributes has the service effect on, and what is the direction of change the service can inflict on the value of the attribute. This additional information helps the DC to select from a pool of services the appropriate one, thus reducing the potential for failure. In the example of fig. 6. this transformation, and description is grasped. We can see that the STYPE-CAN-BOIL-EGG-L2 can modify the values of attributes with name TEMP, and STATE. From the figure it is also apparent that the services and the requirements are connected, and they transfer various objects between each other. It is these objects which the 'real' attributes belong to. In this particular case the object is an EGG, which posses - among others - the attributes TEMP, and STATE. The EGG object is passed to STYPE-CAN-BOIL-EGG-L2 and it carries out its associated transformations on the object, thus altering the values of the object's attributes. The EGG object, in turn, is passed onto the requirement which prescribed the transformation (the service), but when the requirement receives the object, the previous transformations have their effect through the modified attribute values. These objects traverse the system hierarchy, since the complex services may be decomposed and mapped onto lower level requirement. So the whole operation of the system is manifested in a transformational chain of requirements and services.

9. Conclusions and further work

Although the presented ideas and results stem from two basically different fields of research: fault-tolerant systems, and artificial intelligence (AI), and certain overlappings can be observed between the basic goals of these fields, especially in connection with diagnostic and intelligent monitoring systems, there is huge gap between the two areas. AI methods - though offer much more freedom in representing and handling information - stay at the level of "pure" diagnosis, and previous solutions do not tend to cope with fault-tolerant behaviour (see [1,23,24,25,26,27]).

In this article a new modelling method is proposed with the following important potential advantages, which distinguish it from previous solutions for achieving fault tolerance:

- it makes it possible to use implicit redundancy in distributed systems to reach fault tolerant operation;
- the amount of information incorporated in diagnostic centers can be kept at an optimal level, because of the top-down modelling approach;
- there is no need for performing complete diagnostic procedures, which determine the exact causes of faults and normally are very time consuming, so the proposed procedures can concentrate on the fulfilling of the given set of system requirements and so achieving fault tolerant operation, and this way high operating speed can be expected;
- it can host different kinds of data acquisition methods ranging from built-in software exceptions and memory management units through sensors or certain kinds of data acquisition components. Since it does not necessitate traditional, hardware-oriented fault-tolerant solutions, the required hardware overhead can be kept acceptably low;
- it can be applied in systems which were originally not designed to be fault-tolerant.

The proposed model offers the representational and problem solving power to cope with general problems. The implementation of the proposed methodology is supported by both known theoretical concepts and programming tools. A very good summary of these methods is given in [28].

The basic concept of describing different components of the model is using sets of attribute-value pairs. The whole model is constructed by defining *relations* between different components. These properties can be very well provided by a *semantic*

network or a *frame system* [29]. Another important part of model interpretation is *data-flow scheduling* and the handling of *mapping rules*.
Since at the first stage of the implementational work conceptual problems are dominant over real life applications, and run-time efficiency, we have chosen *Common Lisp* as the implementational language for the first experimental version of our system, because Common Lisp seems to offer the most flexible and powerful machinery for solving the tasks mentioned in this article. It is a traditional AI language, so ready made toolsets are available for implementing different rule based systems (which is important for the handling of mapping rules). It has an object oriented extension (CLOS), which directly supports the implementation of frame based systems, so it offers very good capabilities to describe model components. CLOS allows the easy implementation of object oriented data-flow networks, where actor and data nodes are both instances of predefined object classes.

We are working on an experimental implementation, which will be available soon. This will allow us to test the abilities of the approach on various practical benchmark problems, like a simple industrial technology (a water preparation plant), a microwave telecommunication network and a simple, adaptive resource allocation problem in computer networks.

References

[1] S.Hariri, A.Choudhary, B.Sarikaya: Architectural Support for Designing Fault-Tolerant Open Distributed Systems, COMPUTER June 1992, pp. 50-62

[2] R.Benjamins, W.Jansweijer: Toward a Competence Theory of Diagnosis, IEEE EXPERT, Oct 1994, pp. 43-51

[3] M.Polycarpou, A.Helmicki: Automated Fault Detection and Accommodation: A Learning Systems Approach, IEEE Trans. on Systems, Man, and Cybernetics, Vol 25, No 11, Nov 1995, pp. 1447-1458

[4] K.Tilly et al.: A Method for the Construction and Interpretation of High Level Models for Distributed Fault-Tolerant Systems, IEEE Symp. on Reliable Distributed Systems, SRDS-95, Bonn Sept. 1995

[5] K.Tilly, et al.: The Application of Intelligent Diagnostic Centers in the Implementation of Distributed Fault-tolerant Systems, Dept of Measurement and Instrument Eng., Technical University of Budapest, Technical Report TUB-TR-94-EE09

[6] V.Nelson: Fault-Tolerant Computing: Fundamental Concepts, COMPUTER, July 1990, pp. 19-25

[7] F.Cristian: Automatic reconfiguration in the presence of failures, Software Engineering Journal, March 1993, pp. 53-60

[8] W.Heimerdinger, Ch.Weinstock: A Conceptual Framework for System Fault Tolerance, Technical Report CMU/SEI-92-TR-33, ESC-TR-92-33

[9] H.Kopetz, G.Grunsteidl: TTP - A Protocol for Fault-Tolerant Real-Time Systems, COMPUTER, January 1994, pp. 14-23

[10] Y. Umeda et al.: Using Functional Maintenance to Improve Fault Tolerance, IEEE EXPERT, June 1994, pp. 25-31

[11] S. Kobayashi, K. Nakamura: Knowledge Compilation and Refinement for Fault Diagnosis, IEEE EXPERT, Oct 1991, pp. 39-46

[12] Hwee Tou Ng: Model-Based, Multiple-Fault Diagnosis of Dynamic, Continuous Physical Systems, IEEE EXPERT, Dec 1991, pp. 39-43

[13] R. Iserman: Process Fault Detection Based on Modelling and Estimation Methods - A Survey, Automatica, Vol 20, No 4, pp. 387-404, 1984
[14] J. Vinson, et al.: Automatic Rebuilding of Qualitative Models for Diagnosis, IEEE EXPERT, Aug 1992, pp. 23-30
[15] D. Dvorak, B. Kuipers: Process Monitoring and Diagnosis, IEEE EXPERT, June 1991, pp. 67-74
[16] B.Homayoun, M.Nakamichi: Qualitative Fault Diagnosis in Systems with Nonintermittent Concurrent Faults: A Subjective Approach, IEEE Trans. on Systems, Man, and Cybernetics, Vol 23, No 1, Jan/Feb 1993, pp. 14-30
[17] Jong Man Lee, Jin H. Kim: An Integration of Heuristic and Model-based Reasoning in Fault Diagnosis, Engng. Applic. Artif. Intell., Vol 6, No 4, pp. 345-356
[18] A.Aldea, M. Chantler: Time-Constrained Diagnosis Using Hierarchical Models, Proc. of the 1st Int. Conf. on Intelligent System Engineering, 1992, pp. 47-52
[19] Mengly Chean, J. Fortes: A Taxonomy of Reconfiguration Techniques for Fault-Tolerant Processor Arrays, COMPUTER, Jan 1990, pp. 55-69
[20] M. Ancona, et al.: A System Architecture for Fault Tolerance in Concurrent Software, COMPUTER, Oct 1990, pp. 23-32
[21] R. Geist, Kishor Trivedi: Reliability Estimation of Fault-Tolerant Systems: Tools and Techniques, COMPUTER, July 1990, pp. 52-61
[22] D. Siewiorek: Fault Tolerance in Commercial Computers, COMPUTER, July 1990, pp. 26-37
[23] C. R. Kime (1985): System diagnosis. In Fault Tolerant Computing:Theory and Techniques, Prentice-Hall, New York, 1985, pp. 577-623
[24] J. Sztipánovics: Toward structural structural adaptivity, Proc. of ISCAS 1988, Vol. III, pp. 2359-2362
[25] J. A. Carrasco: Hierarchical object orientedmodelling of fault tolerant computer systems, Proc. of COMPEURO 1991, pp. 452-456
[26] J. Crow et al.: Model-Based Reconfiguration: Toward an Integration with diagnosis, Proc. of AAAI 1991, pp. 836-841
[27] R. Reiter: A theory of diagnosis from first principles, Artificial Intelligence, Vol. 32, pp.57-95
[28] E. Rich et al.: Artificial Intelligence, McGraw Hill, 1992
[29] M. Minsky: A framework for representing knowledge, The Psychology of Computer Vision, P. Winston (Ed.), McGraw Hill, New York, 1975