Matching requirements to resources -
A Step Toward Flexible Fault Tolerant Distributed Systems[1]

Gy. Román, A. R. Várkonyi-Kóczy, T. P. Dobrowiecki, K. Tilly
Technical University of Budapest
Department of Measurement and Instrumentation Engineering
H-1521 Budapest, Mûegyetem rkp. 9.
E-mail: roman@mmt.bme.hu

1. Introduction

With the spread of computers and the revolutionary development of network facilities highly distributed systems evolved. Such systems make it possible to reconsider the idea of fault tolerance at a higher level of abstraction. Distributed architecture leads directly to distributed diagnosis, and generally speaking, it has much more potential than what is purposefully utilised. In recent applications systems are highly sophisticated, typically containing multi-functional components. These components directly realise functional redundancy, i.e. the means of improving the fault tolerant behaviour. In addition, the role of computing elements drastically increased, and such systems offer extra computing capacity to be exploited for diagnostic purposes.

When supervising a complex system, the descriptive power of the model grasping the operation, structure of the system, and the potentials of its components is of utmost importance. Though diagnosing the faulty components is crucial, the overall goal of fault tolerant systems is to perform error-free operation even in the presence of some faults. To cope with the increased complexity of distributed systems, the authors propose a novel modelling approach with emphasis on the fault tolerant aspects of the operation. The essence of the proposal is the fault tolerant behaviour based upon the functional substitution [6,7] of the faulty resources. To this end, redundancy at the intelligence level is introduced into the system. Intelligent diagnostic centers at different levels of system hierarchy map various resources and redundancy to the possible failures, and infer the best possible functional substitution of the failed resources.

The approach addresses the issue of system modelling at a functional level. It seeks means to express the functional decomposition of a system [6, 7] into a suitable hierarchy, using primary concepts of '*service*' and '*requirement*'. Requirements represent the prescribed (sub)goals, services represent the possible means. To describe both primitives, the concept of '*attribute*' is introduced. Based on this description, services that can satisfy certain requirements can be modelled and dynamically sought out within the functional hierarchy of the system description.

To support this search efficient algorithms are needed that can cope with the dynamic behaviour of system components. The authors examine the tasks and problems that arise in connection with selecting appropriate services to satisfy requirements, and propose a dynamic attribute matching algorithm for efficiently finding the mapping between services and requirements.

---

[1]This work was supported by the Hungarian Scientific Grant Nr. T-014403.

2. The modelling approach

Traditional fault tolerant systems [4, 5] are fault tolerant, because they were designed to be. During the design phase developers intentionally introduce redundant components which, should a fault occur, can take over the task of the faulty component. However, the redundant components add to the cost of the system, though in several applications reliability is more important than costs.

On the other hand, there are a lot of systems which originally have not been designed to be reliable, or in the meantime, the expectations in reliability have increased. One way of coping with this problem is to redesign the system, which, however, is not a feasible solution at least from economical point of view. Another solution can be to add certain components to the system, which, in turn (when a very important component becomes faulty) can substitute the faulty component.

Given a closer look to recent complex systems (see fig. 1.), it becomes transparent that these systems contain much more computational elements than previous systems; and in many cases a considerable computational capacity is not used during operation. From the above it naturally follows that the free computational power can be exploited to support fault tolerant behaviour. In order to be able to utilise this capacity, the operation, and the structure of the system have to be reconsidered, and a full functional model has to be formalised. The formal description has to be extended to the individual components of the system, explicitly expressing all their potential, thus making it possible to use the redundancy naturally present in the system.

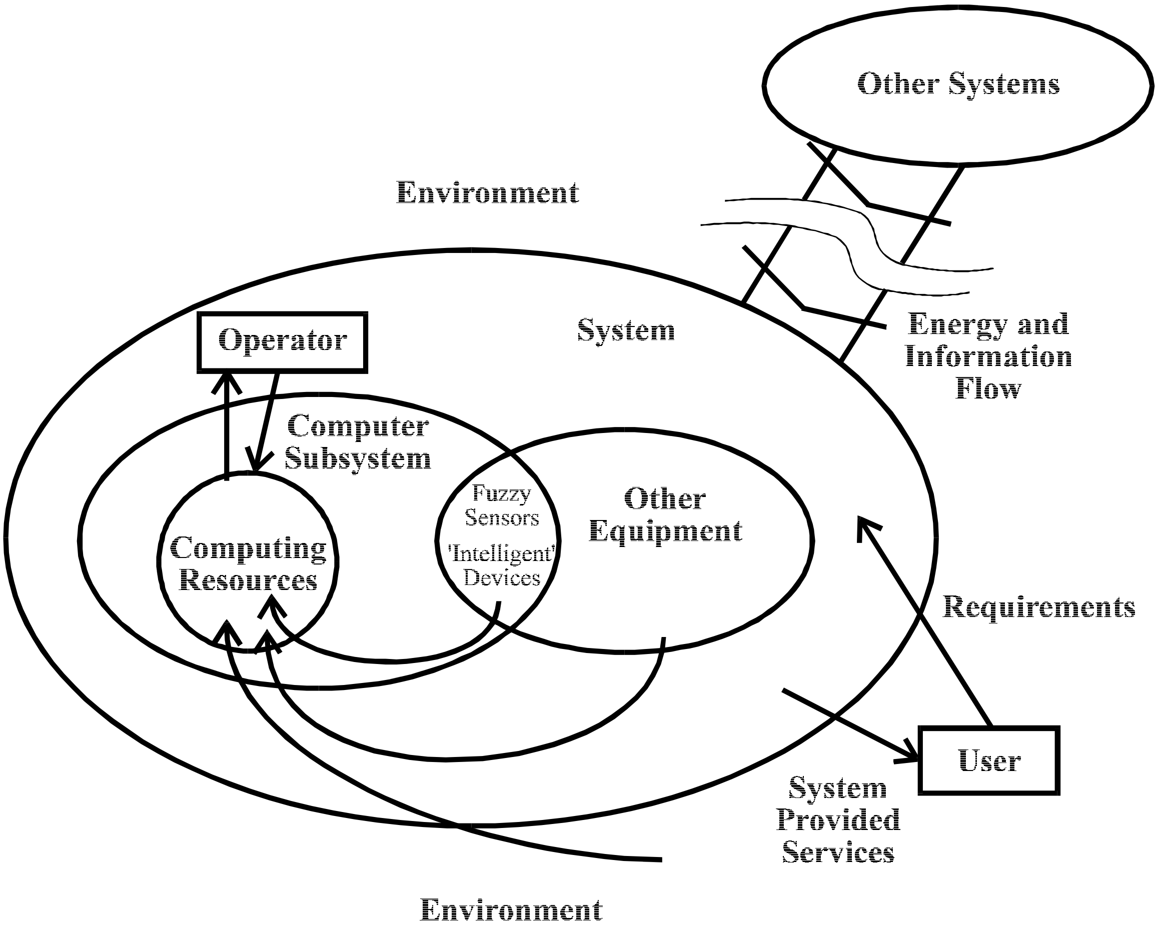

Figure 1.: The schematics of a complex industrial system

In order to efficiently utilise the redundant computational capacity, a simple, but expressive model structure is proposed (see fig. 2.). The basic idea is that the system's operation is functionally modelled; and to express the (sub)goals and the means to achieve them two notions: requirements and services have been introduced respectively. The idea is simple: every task, being complex or simple, is formally described by a requirement, which states functionally what has to be done. On the other hand, all the capabilities of the system (of its components) are described by services. Services can also be of arbitrary complexity, the only restriction is that there has to be some means upon which to decide whether the service has been successful or not. (In industrial systems these 'means' can be e.g. some sensors.)

When the system is operating normally, there is an active mapping from all the requirements to some service, and this mapping constitutes a service-requirement network, giving the formal decomposition of the problem.

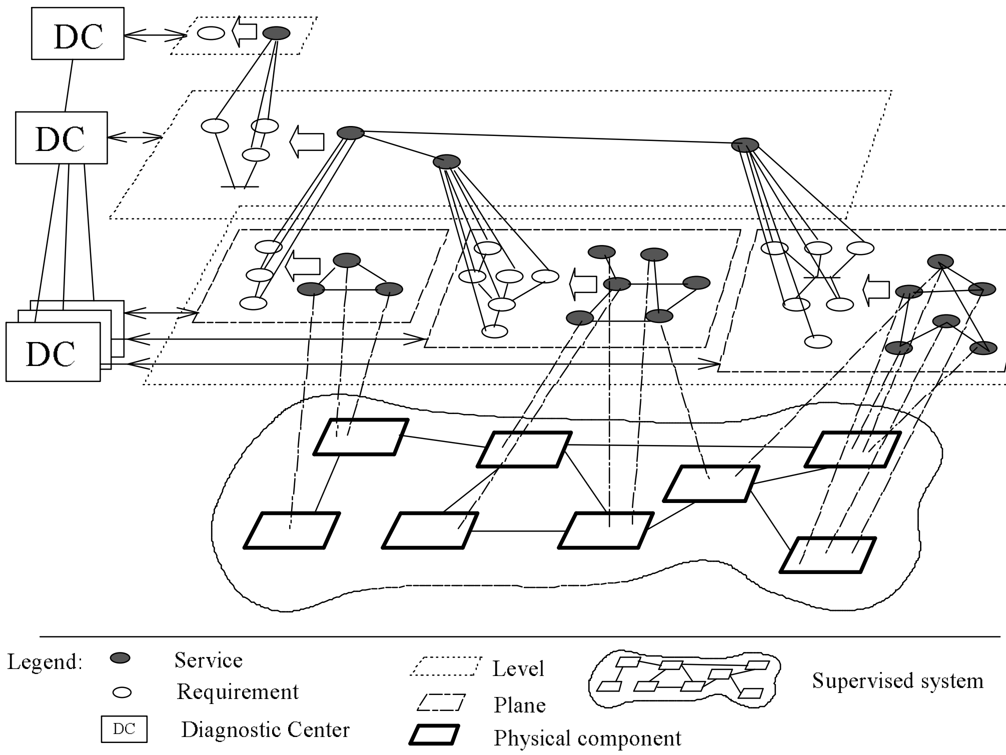

Figure 2.: The architecture of the diagnostic system

Another important feature of the architecture that it is hierarchically decomposed into different levels of abstractions. If we examine fig. 2. more carefully, we can see that all the requirements are mapped onto - and thus are satisfied by - services on the same level, while services can be decomposed to lower level requirements. This means that the operation, the overall goal of the whole system is expressed by a requirement (or a set of requirements) at the topmost level, and at this very same level an abstract service that is capable of solving the problem on this level of abstraction is given. However, in many cases, this top level service is really an abstraction, no physical service (equipment, agent) can be associated to it. In such cases the service of high abstraction is decomposed to lower level requirements, which, in turn are mapped onto services on the same level, thus decomposing the higher level service to lower level services. Thus, requirements are not only there to express the goals, but through them simple services can be combined into more complex services.

Now let us examine the model components in more detail.

3. The definition of the model

The proposed model decomposes the problem into *levels*, representing abstractions of the *whole* system. The levels are built of *planes*, where the model has a given structure of *components* (defined in the following sections), and planes are connected to each other in a well defined way [2, 3].

A diagnostic center is associated to every plane, in order to observe and maintain its services toward 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), while information on compound components (or simply *compounds*, having a decomposition at another plane) is delivered by the diagnostic centers associated to the lower level plane holding the decomposition.

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.
- *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 capable of providing different types of services. Resources can be *physical* (like a computer hardware) or *conceptual* (like a software module). 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 *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. Thus, the attributes play a key role in the efficient operation of the system.

Before turning our attention to the problems associated to attribute matching, let us consider one of the above discussed model elements - the services - in more detail.

4. Services revised

In the previous section we have seen that the services are means to fulfil requirements. However, based on their roles, they can be categorised into two groups:

- provider services; and
- transformer services.

This distinction is very important, since the services act on real-world objects, and some means is needed to coherently couple these objects with the model. Authors chose a simple but rather powerful approach: a class of 'primitive services' is introduced, which contains those services, whose primary function is to provide a certain type of object (though they usually also perform some transformations).

Since the real world objects display well distinguishable properties, the provider service class is further divided into three categories:

1. countable;
2. uncountable; and
3. reusable.

The first category describes services, where the object to which the service is related, is countable, and the resource providing this service possesses a certain piece of that service (and thus that object). (Taking example from the world of egg boiling, this type of service can be that we have some eggs - CAN-FIND-EGG.) After providing the service the resource will be capable of providing one less from that type of service. In the second case, the object can be described with some quantity, and providing services of this type decreases the quantity of the object in the resource. (For example, water in a jug can be such an object, and CAN-FIND-WATER can be a service that provides a certain amount of water.

After providing the service, the amount of water in the resource would decrease with a certain amount.) The resource is capable of providing the service till the amount of the object is greater than the amount requested by the service. The common feature of these two service types that they are not reusable. Once they were provided, they leave the resource in a modified state, which will not be restored after the completition of the service (though the resource state can be restored with other services). The third category of services - the reusables, describe those services which for the duration of the service will allocate some of the objects of their resource, but after the completition of the service they would return the used object to the resource, thus restoring it to its previous state. With these provider services the objects from the environment can be coupled into the model (through the resources).

Since services modify certain attributes of their input-objects, in the representation of services it has to be explicitly enumerated that which attributes they have effect on. This information gives support for the diagnostic center in selecting the appropriate service to fulfil a given requirement. Furthermore, the direction of the modification is also given, thus a more detailed information is available for the DC, upon which to base its decision. (The above direction tells the DC that the given service can increase, or decrease the value of an attribute, or in the case of enumeration type attributes, it gives the possible value transitions.)

On the other hand, since time plays central role in dynamic systems, time-dependant properties of services also have to be modelled. Figure 3. shows a simple transformation service STYP-CAN-HEAT, whose function is to raise the temperature of any object going through it. Since this service is a dynamic one, some means to describe its time-dependant behaviour is needed. The effect of the service on the object is given by a function of two variables, one variable being time, the other being the input-object. Based upon this function a preliminary time value can be predicted which determine the approximate completition time of the given service. In the example of fig. 3. STYP-CAN-HEAT has effect on the object's temperature. This service is used to bridge the gap between the temperature of an object and a requirement's temperature prescription. In this case the input object is an egg with temperature 5 °C, and the requirement is an egg with 35 °C.

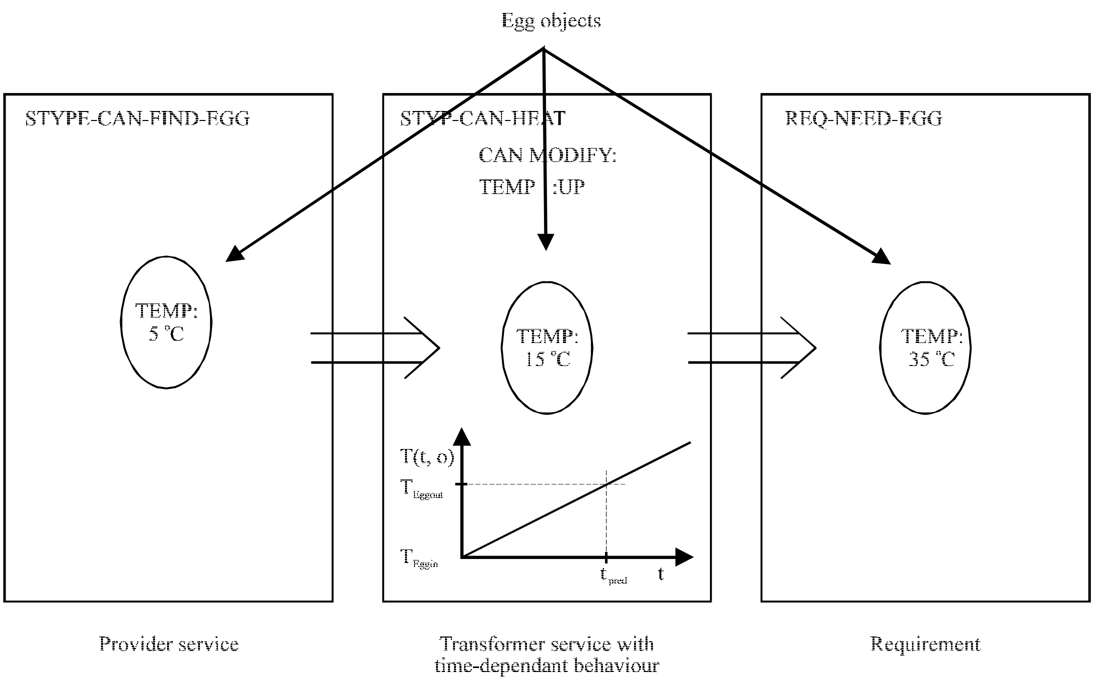

Figure 3.: The dynamic properties of services

The information provided by the predicted time-point can be exploited by the diagnostic center associated to the plane of the given service to check the completition of the service. (This is meaningful when the service is not continuously monitored. On the other hand, if it is continuously monitored, it allows for the early detection of a faulty service. In this case, we have the information that service

STYP-CAN-HEAT possesses a monotonically increasing temperature characteristics. If the DC finds out that the measured characteristics is monotonically decreasing (see fig. 4.), it can deduce that there is no chance of completing the heating service. Furthermore, it is possible to compare the theoretical characteristics with the measured one; and based on the monitored values, the predicted service completition time-point can be recalculated.

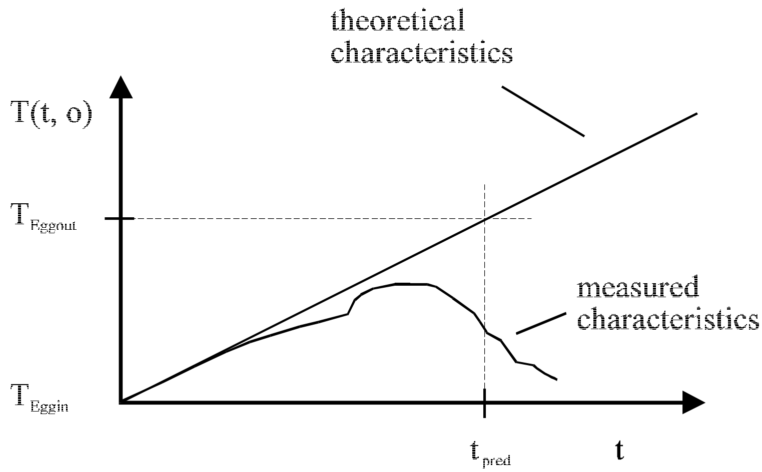

Figure 4.: Example of a faulty time-dependant service

Since sometimes a given object has to go through a chain of service-requirement connections, some means to explicitly describe the path of the object in the service-requirement network is needed. The approach builds on the concept of input-, and output-objects. Every service and requirement possesses slots to describe the actual input-objects, and output-objects. (In case of services the service performs its characteristic transformations, thus changing some of the attributes of some of its input-objects.) Since during the operation of the system there are active mappings between services and requirements, the idea to define the path of objects is straightforward: the output-objects of a service are directly mapped onto the input-objects of the corresponding requirement. (When the service handles only one input-object, and it transforms its attributes, the interpretation of the transformation network can be visualised with a parallelism: we can imagine the whole network as a subway system, the lines representing the service-requirement connections, and the stations representing the services, where transformations are carried out on the objects. Thus, the operation of the system is nothing else than the objects are moving in the tunnels, as trains, going towards the end-station (representing the overall goal of the system), while at every station they undergo some modifications.)

Figure 5. shows an example of how an object goes through transformer services to meet the requirements. The object starts from a lower level requirement, it enters the service by its input-objects slot, it passes through the transformer service (during which some of its attribute values are modified), and since the output-objects slot of the service is connected to the input-objects slot of the related through it the object can pass on to higher level services.

The provider services fill in a gap in this description: some kind of objects are attached to their input-objects slot, thus starting the movement of objects (see fig. 4.).

Usually, when a requirement is formalised, the existence of an object (or group of objects) is prescribed, with certain properties. (The type of object and its properties are equally important.) Since the properties are formally expressed by attributes, and with their values, the requirements contain prescriptions on the values of certain attributes (and, of course, on their existence).

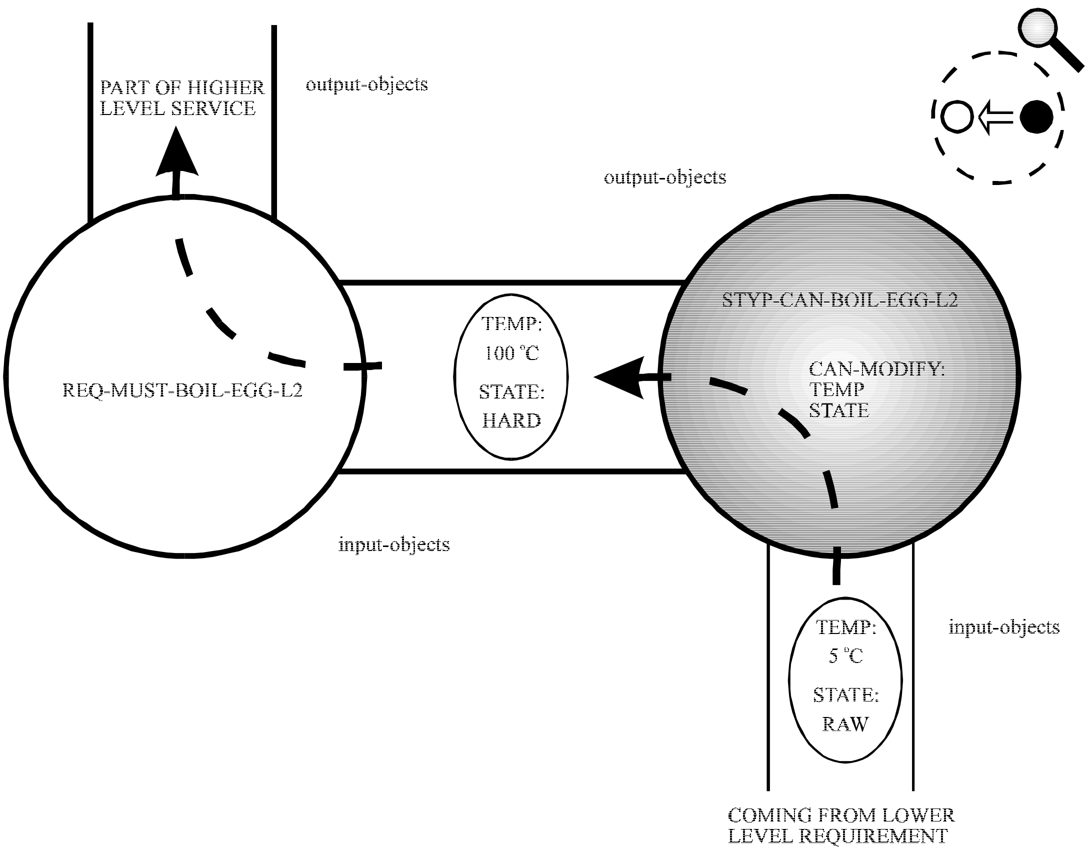

Figure 5.: The transformation of object attributes: the requirement-service network

5. Attribute matching

Before going into the details of the attribute matching algorithm, first let us see what kind of attributes the system handles. Based upon the possible values that an attribute can take, they are classified into three categories:

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

These simply mean that the attributes of given type can have a certain value, a value from an enumerated list, or an interval as their value respectively.

In connection with the value and interval type attributes some measure of the matching can be calculated and with its help the quality of match can be expressed. This can be exploited when optimal performance is aimed at.

In the previous section we examined the services from the point of view of their dynamic behaviour. When the system performs time-dependant behaviour, it imposes serious requirements on the attribute matching algorithm.

If time is eliminated from the model, attribute matching also can be carried out in a static way: mappings have to be searched for, where the attribute values in the requirement and in the given service exactly match each other. However, as soon as the duration of completing a service is also integrated into the model, the above static matching is not appropriate. Since some services carry out transformations on objects, thus modifying some of its attributes, not exact matches, rather possible matches have to be identified. In this case it is not sure that if a given attribute do not match it will never match. As we have seen in the previous section, in the services it is explicitly enumerated that which attribute values can a given service modify; furthermore, the direction of that change is also given. With this information services that possibly transform the object's not matching attributes towards matching can be selected.

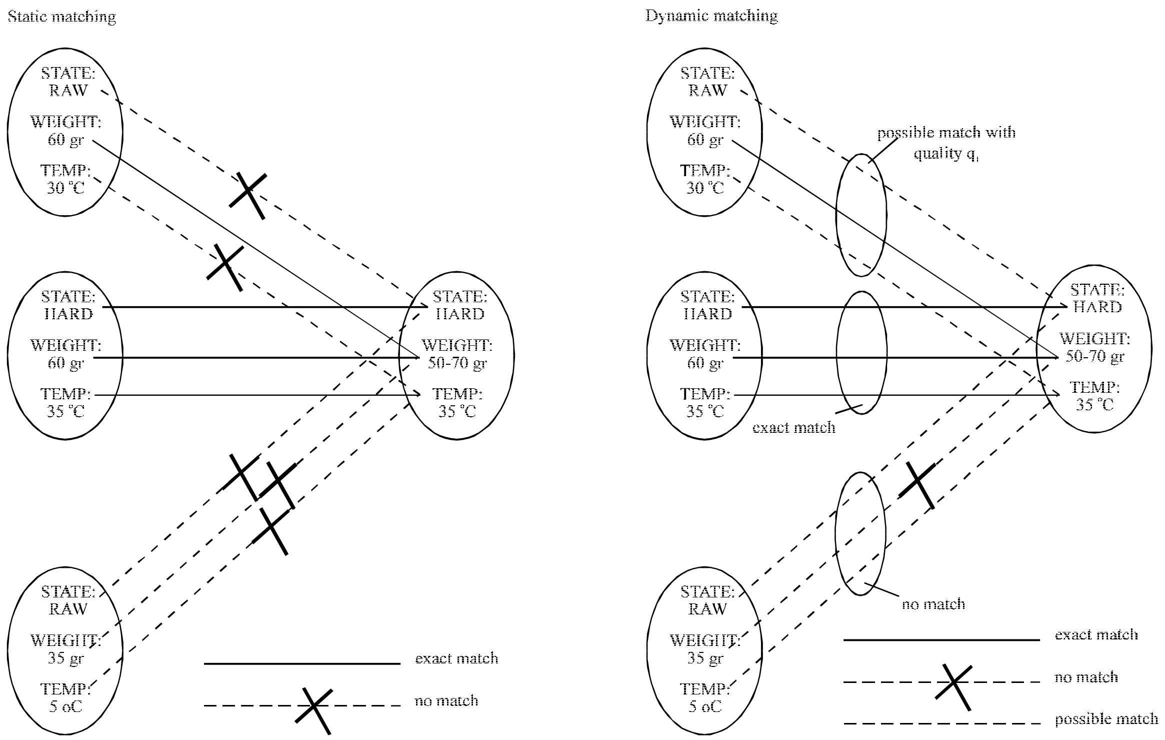

Figure 6.: The difference between static and dynamic attribute matching

This draws attention on the quality of the match. While in case of static matching we speak only of exact, or perfect match, in the dynamic case we very often have to be satisfied with possible match (see fig. 6.). If it turns out that, at the end, the match was unsuccessful, it is the diagnostic center's task to find another possible matching service (a service that may help in satisfying some requirement).

6. Conclusions and further work

The present work tries to merge methods of two different research fields - artificial intelligence (AI) and fault-tolerant systems - into a single, coherent body: it proposes a framework that can be efficiently utilised for making systems with extra computational power more fault-tolerant. 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 [1,8,9,10,11,12]).

In order to be able to exploit the representational and problem solving power offered by the presented model, a dynamic matching algorithm has to be utilised. For

The modelling approach builds on the dynamic attribute matching algorithm. The implementation of the model and the algorithm is done in Common Lisp, which is a traditional AI language. Its object oriented extension (CLOS), directly supports the implementation of frame based systems [13, 14], so it offers very good capabilities to describe model components; and it also allows the easy implementation of object oriented data-flow networks. The experimental implementation, which will be available soon, 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.

Though the concept is grasped by the implementation, there are further theoretical problems to examine in more details. Though the architecture itself bears no restriction on the concurrent implementation of the diagnostic centers and the planes, there are numerous practical problems to cope with in connection with the matching algorithm. On the other hand, when strict real-time requirements are imposed on the operation of the system, the anytime algorithms [15] might be of help. Some of the services ab ovo display anytime behaviour, and it is straightforward to implement them that way. Other services are not appropriate by their very nature to be transformed into an anytime algorithm, but there is a third class of services, which by some means, can be transformed. The effect of the use of anytime algorithms on the attribute matching is the subject of further examination.

A very important aspect of real problem solving is that, based on some criterion, it is very often meaningful to discriminate between the solutions. Stemming from the natural redundancy present in complex systems, usually the very same service can be carried out in more than one way. These services bear different costs depending on the criterion used. It is crucial that during the search for the dynamic mapping between requirements and services, the optimal mappings have to be found. A detailed examination is needed as how to formally integrate the optimum criteria into the system; and how to combine partial criteria into one global criterion.

References

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

[2] 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

[3] 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

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

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

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

[7] 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

[8] C. R. Kime (1985): System diagnosis. In Fault Tolerant Computing:Theory and Techniques, Prentice-Hall, New York, 1985, pp. 577-623

[9] J. Sztipánovics: Toward structural structural adaptivity, Proc. of ISCAS 1988, Vol. III, pp. 2359-2362

[10] J. A. Carrasco: Hierarchical object orientedmodelling of fault tolerant computer systems, Proc. of COMPEURO 1991, pp. 452-456

[11] J. Crow et al.: Model-Based Reconfiguration: Toward an Integration with diagnosis, Proc. of AAAI 1991, pp. 836-841

[12] R. Reiter: A theory of diagnosis from first principles, Artificial Intelligence, Vol. 32, pp.57-95

[13] E. Rich et al.: Artificial Intelligence, McGraw Hill, 1992

[14] M. Minsky: A framework for representing knowledge, The Psychology of Computer Vision, P. Winston (Ed.), McGraw Hill, New York, 1975

[15] S. Zilberstein: Operational Rationality through Compilation of Anytime Algorithms. PhD Dissertation, Computer Science Division, University of California, Berkeley, 1993.