1 Introduction

Fault tolerant systems are designed to hide the effects of specific (anticipated) faults. If the system is specified and designed in a formal environment, behavioral equivalences can be used to examine the relationship between the ideal, fault-free behavior and the behavior in the presence of anticipated faults. [1] proposed a framework using LOTOS as the specification language and observational equivalence, which relates two systems to be equivalent if an external observer cannot distinguish between them.

In our work, we adopted this framework to UML (Unified Modeling Language [5]) as a specification and design language. To do this, at first extensions of the UML were necessary to define the relevant information which is to be compared. The next step was the elaboration of the transformation from UML behavioral diagrams to labeled transition systems on which equivalences can be checked by available tools. These steps will be described in the following sections.

2 The UML framework

UML is the most recently created modeling language of object-oriented systems. It comprises a variety of diagrams adapted to the different views of the system used by a practitioner designer in the (functional) design process. Since we are interested in the behavior of the system, besides the static structure (describing the classes and objects) the behavioral diagrams are interesting. Behavior of objects (classes) is described by a variant of statecharts. In statecharts, states can be refined hierarchically supporting also internal concurrency. State transitions are labeled by trigger events, guards and actions. The behavior is specified as a traversal of the graph of states along the transitions, triggered by events resulting from actions or changes in the system or in its environment. Events are processed sequentially, one by one.

The externally observable behavior is given by the sequence of trigger events (which can be captured by the system) and actions (which are available for the environment). In order to keep the complexity of the verification at an acceptable level, usually not complete systems, but subsystems responsible for the fault tolerant behavior (e.g. redundancy managers, fault tolerant controllers, or a package of redundant objects) are examined. Often the environment of these subsystems is not completely defined in the early phases of the design. Accordingly, we offer the designer the possibility to designate the interface of the subsystem through which the behavior is observed and to indicate what are the events and actions "interesting" for the outside world. In UML, structuring is supported by built-in features like interfaces, packages and inherent encapsulation. Renaming and hiding is supported by standard extensions like stereotypes (e.g. transitions with stereotype <<hidden>> will not be considered in the behavior) or constraints.

In the current phase of our work, the behavior in the presence of faults should be described explicitly by the designer, using the standard model elements of statechart diagrams. UML constraints can be used to restrict the behavior, e.g. to limit the number or type of faults.

3 The transformation

Behavioral equivalences are defined on labeled transition systems (LTS), a formalism based on the primitive notions of states and transitions. Transitions are labeled with actions. We consider non-deterministic LTS being capable of performing internal transitions (usually labeled by τ) which can not be seen by external observers.

The behavioral descriptions of the two subsystems to be compared have to be transformed to two LTSs. This transformation is based on the formal operational semantics of UML statecharts defined in [4] using Kripke structures. The resulting LTS defines the possible sequences of internal transitions, observable UML events and actions.

The transformation was implemented from the model repository of the UML tool Innovator [6] to the input format of the Aldebaran tool set [2]. As an intermediate step, in our experimental implementation we described the statecharts in Promela, input language of the model checker SPIN [3]. SPIN was used to dump the global states and transitions of the system.

The Aldebaran tool set supports efficient checking of observational equivalence but also various other behavioral equivalences and preorders. The hidden and renamed actions can be given as external constraints, this way these definitions could be extracted from the
UML diagrams separately. The hidden actions are represented as internal transitions in the LTS.

### 4 An example

Our work is illustrated by an example. An object broker is used to forward the request of a client to a server. The requests of the client fall into two categories: critical requests (creq) which should always be served (cresp) and non-critical requests (nreq) which can be served (nresp) or refused by the server (reporting nfail to the client). All other events and actions are hidden from the client. In the ideal case, the behavior of the broker is depicted in Figure 1. For the sake of simplicity, the destination objects of actions are not given.

![Figure 1. Statechart of the simple broker](image)

The server can fail in a fail-silent manner (there is no response), in this case the requests are not served. In order to increase the availability of the service, the object broker is modified by utilizing two more objects, a watchdog timer and a safety core (Figure 2). The broker manages these additional resources in the following way: The failure of the server is detected by the watchdog timer. In the case of non-critical requests, the failure of the server is handled by recovery and retry (in this way transient faults can be tolerated). In the case of critical requests, the safety core, which implements only the critical functions, will provide the service (tolerating both transient and permanent faults of the server by active redundancy). The behavior of the modified broker (including the behavior in the presence of the anticipated faults) is depicted by the statechart of Figure 3.

![Figure 2. Class diagram of the broker](image)

![Figure 3. Statechart of the broker](image)

### 5 Conclusion

We presented an adaptation of the framework proposed for formal reasoning on fault tolerant techniques in [1]. In the object-oriented environment of UML, examination of behavioral equivalences can be used not only for the verification of the fault tolerance coverage, but also to check consistency of model refinement (conformance to initial specification) and inheritance (subtyping).

### References