1. Introduction

More and more, monolithic systems are giving place to systems built from interconnected systems, designed by independent parties. These systems employ communication standards to interact. Sometimes, one of the systems does not implement the standard correctly, or implements it only partially. This may occur, for example, if the specification of the standard was misinterpreted, or if there was insufficient debugging or testing. In those situations, interaction faults may occur when the systems try to communicate.

In this paper, a new scenario for tolerating interaction faults is presented. We also address the problem of designing a system capable of tolerating interaction faults generated by the other system. The scenario and other concepts defined in this paper were discussed with more detail in [3].

The remainder of this paper is organized as follows. In section 2, the system and fault models are defined. In section 3, we describe the general schemes for error detection and correction. Section 4 concludes the paper.

2. System and Fault Models

A system is defined as a pair of sub-systems that use a communication standard to interact. Interaction occurs with the exchange of a sequence of messages, each containing a set of data fields. The system that can exhibit faulty behavior is called the external unit (E). The other system, which is designed to tolerate faults, is the adaptable unit (A).

To manage the complexity of the standard, it is divided in tasks. A task is a group of states working towards a single purpose in the standard, such as as “end connection” or “request file”. Each task is defined by its semantics, the type of information it deals with, and a communication pattern. Communication patterns describe sequences of messages, and may be represented by regular expressions.

To isolate the problem, it is assumed that, apart from the implementation of the standard, the external unit is fault-free. That includes lower-level protocols needed by the considered standard, which are abstracted as a fault-free “communication interface”. We also assume that all faults originate from E, not from the unit being designed (A).

Using the classes presented in [1], we consider non-malicious, non-deliberate interaction faults that result from human errors. Even though they are caused by a flawed protocol implementation, they are not design faults, as they occur in operational time. From E’s point of view, they are design failures, but for A, they are interaction faults. The faults are also permanent — if E is repaired, we consider a new system instance.

When a fault occurs, it may result in an error, modeled based on how it is observed outside E. The errors are divided in classes:

- **Incorrect Action**: E performs an unexpected action.
- **Incorrect Data**: is divided in three sub-categories:
  - Missing Data: a field inside a message is missing.
  - Obviously Incorrect Data: the value for a field is unacceptable for its type.
  - Not Obviously Incorrect Data: a field has an acceptable, but incorrect, value.
- **Formatting**: E sends an ill-formatted message.
- **Sequence**: E sends messages or perform actions out of the expected sequence.
- **Timing**: E performs the expected actions and sends eventual messages, but too early or too late.
- **Omission**: E does not send a message it was expected to send, or sends it with a very long delay.

3. Detecting and Correcting Errors

The adopted system model determines and limits the mechanisms that can be used for error detection and correction. Three fundamental peculiarities can be described:
• The presence of an external unit. As $E$ is not part of the unit being designed, $A$’s designer has no control over the faulty implementation of the standard. This scenario is different, for example, from conformance and interoperability testing [2], which focus on detecting design faults in the system under development. Moreover, the top-level system is only instantiated when the units communicate — i.e. the specific external unit is not known while $A$ is being designed, and error detection and correction must be done online.

• The communication standard has been already specified. That means no changes can be made to the protocol, as $E$ cannot be required to implement them.

• There is only a pair of units. That means $A$ cannot rely on other units besides itself to detect and correct errors.

Given these restrictions, error detection and correction mechanisms are mostly based on knowledge about the standard and the problem domain. Of special interest is the notion of implicit redundancies. We define implicit redundancies as information which is not originally used to support fault tolerance, but is somehow duplicated in the protocol, or gives a hint about what can be expected in the future. This “entropic” information should be saved after it is received, as it can be employed later to detect and correct errors.

Below, we describe the general mechanisms that can be used to detect each type of error.

• Timing and omission errors are traditionally detected by timers, with larger timeouts for omission errors.

• Sequence errors can be detected by comparing the order of the messages with the communication pattern, or by checking fields such as sequence numbers.

• Formatting errors can be detected by traditional format parsers, as message formats can be described by context-free grammars.

• Missing data errors can be detected by checking for the presence of required data fields inside the message.

• Other incorrect data errors are detected with comparisons with “references”. These references are different for each type of data: they can be a set of if/then statements, range checks, mathematical functions, etc. They may incorporate information coming from the protocol specification, the designer’s knowledge about the problem domain and implicit redundancies.

• Incorrect action errors may result in other types of error — being detected that way. If this is not the case, they may be only detected if they result in very specific patterns of behavior, such as cycles or repetitions.

Some timing errors can be corrected by changing the expected timeout for incoming messages. Some sequence errors can be corrected by holding messages until the order is re-established by the receiving of other messages. Correcting other types of errors involve choosing a course of action. One course of action is replacing the missing or incorrect data with information obtained from implicit redundancies, or other “harmless” information (such as default values). Other courses of action include activating other tasks and sending requests with different parameters. The best course of action can only be decided based on the specific task and the context the error occurred.

4. Conclusion

In this paper, we described a new scenario for fault tolerance considering interaction faults. We addressed the problem of creating a system able to detect and correct errors originated from an external system. The greatest obstacle for creating detailed general mechanisms for fault tolerance is that some types of errors depend heavily on the semantics of the tasks, messages and fields. Therefore, this problem should be described following a general model, but must be addressed in a case-by-case manner.

To the authors’ knowledge, this problem is essentially new. Existing work address the design of protocols, aiming at specific applications. Our scenario provides a more general and abstract model, which enables new fault tolerance schemes to be added to the adaptable unit, besides the ones defined by the protocol. Furthermore, our model works under the assumption that the communication standard may not be correctly implemented by one of the units — a real problem which is hardly into account by protocol designers. This scenario, including new concepts such as the implicit redundancies, was discussed with more detail in [3].

The proposed models must be further explored, being employed in a specific case. This will enable us to create detailed error detection and correction mechanisms, and explore the concept of implicit redundancies.

Acknowledgement: This work was motivated in part through discussions in the collaboration with Matsushita Electric Industrial Co., Ltd.

References

