Solving Consensus Using Structural Failure Models

Timo Warns*
Carl von Ossietzky University of Oldenburg
Graduate School TrustSoft
26111 Oldenburg, Germany
timo.warns@informatik.uni-oldenburg.de

Felix C. Freiling
University of Mannheim
Chair of Practical Computer Science I
68131 Mannheim, Germany
freiling@informatik.uni-mannheim.de

Wilhelm Hasselbring
Carl von Ossietzky University of Oldenburg
Software Engineering Group
26111 Oldenburg, Germany
hasselbring@informatik.uni-oldenburg.de

Abstract

Failure models characterise the expected component failures in fault-tolerant computing. In the context of distributed systems, a failure model usually consists of two parts: a functional *part specifying in what way individual processing entities may fail and a* structural *part specifying the potential scope of failures within the system. Such models must be expressive enough to cover all relevant practical situations, but must also be simple enough to allow uncomplicated reasoning about fault-tolerant algorithms. Usually, an increase in expressiveness complicates formal reasoning, but enables more accurate models that allow to improve the assumption coverage and resilience of solutions.*

In this paper, we introduce the structural failure model class DiDep *that allows to specify directed dependent failures, which, for example, occur in the area of intrusion tolerance and security. DiDep is a generalisation of previous classes for undirected dependent failures, namely the* general adversary structures*, the* fail-prone systems*, and the* core and survivor sets*, which we show to be equivalent. We show that the increase in expressiveness of DiDep does not significantly penalise the simplicity of corresponding models by giving an algorithm that transforms any Consensus algorithm for undirected dependent failures into a Consensus algorithm for a DiDep model. We characterise the improved resilience obtained with DiDep and show that certain models even allow to circumvent the famous FLP impossibility result.*

*This work is supported by the German Research Foundation (DFG), grant GRK 1076/1.

1. Introduction

Limitations of threshold models. In fault-tolerant distributed computing, the most commonly used structural failure models are *threshold models*. For example, the SIFT project [20] was one of the first projects that used them with fault-tolerant algorithms. Threshold models characterise the sets of failed components by t*-out-of-*n *assumptions*, that is, at most t out of n components may fail simultaneously. The major advantage of these models is their simplicity that eases proofs of correctness. However, it is well known [13] today that they have strong limitations drawing them inapplicable for many practical situations. A major limitation is the implicit assumption that failures are stochastically independent.

One domain in which threshold models are particularly unsuitable is the area of *intrusion tolerance*, that is, fault tolerance extended to the domain of secure systems. The idea of intrusion tolerance is intriguing: By regarding attacks as faults, classical fault-tolerant architectures and algorithms can be applied in security-related settings [18]. Indeed, the effects of an attack like the infamous "ping of death" [2] are similar to a classical hardware crash and can often be modelled using the same functional failure model. However, if a system consists of identical servers and one of them is vulnerable to the ping of death, all of them are vulnerable and fail *dependently* since an adversary deliberately coordinates his attacks. Threshold models cannot describe such situations accurately such that algorithms developed using a threshold model may suffer from low assumption coverage if used in intrusion-tolerant systems.

25th IEEE Symposium on Reliable Distributed Systems (SRDS'06)
0-7695-2677-2/06 $20.00 © 2006 IEEE

IEEE COMPUTER SOCIETY

Undirected dependent failure models. Several refined failure models have been introduced covering dependent failures: *general adversary structures* [8], *fail-prone systems* [15], and *core and survivor sets* [10, 11]. For example, these models allow to specify that all computers running Microsoft Windows or all computers running GNU/Linux may simultaneously fail, but computers of different types do not fail simultaneously.

The refined failure models still do not allow to cover all practical situations, because they only cover *undirected* dependent failures. However, failures among components may be *directed* dependent. For example, *trust* relations among components induce directed dependencies among component failures. Consider a client-server system with one server and a set of clients. Assume that the clients trust the server, but the server does not trust the clients. If an adversary is able to break into the server and the clients download and execute program code from the server, an adversary may easily break into the clients as well. However, he may not be able to break into the server if he only breaks into a client, because the server does not trust the client and checks its requests thoroughly. This scenario cannot be modelled using existing failure models, because the failures of server and clients are directed dependent.

Contribution In this paper, we make the following contributions:

1. We show that the existing classes of failure models for undirected dependent failures are equivalent regarding their expressiveness. This means that, for example, an algorithm for general adversary structures can be transformed into an algorithm for core and survivor sets and vice versa.

2. We introduce the failure model class *DiDep* for directed dependent failures. We show that previous classes are strictly less expressive than DiDep and that it allows to generalise results obtained with previous classes.

3. For the important problem of *Consensus*, we show how to transform any algorithm for undirected dependent failures into an algorithm for directed dependent failures. The idea is to restrict the operation of the former algorithm to that part of the system that only contains undirected dependent failures. Processes which fail dependently are then informed a posteriori about the Consensus decision. This opens the large body of existing algorithms to the DiDep world.

4. As an interesting corollary, we present yet another way to circumvent the FLP impossibility results [6] by showing that certain DiDep models allow to solve fault-tolerant asynchronous Consensus deterministically without any additional tools like failure detectors. This shows that DiDep allows to improve the resilience of fault-tolerant Consensus.

Roadmap. The paper is organised as follows. First, we present the system model and introduce the notion of functional and structural failure models in Section 2. In Section 3, we describe previous classes of structural failure models and prove that previous classes for undirected dependent failures are equivalent. In Section 4, we introduce our DiDep class and show that it allows to circumvent the impossibility results of Fischer et al. [6] for certain models. We give a general transformer for Consensus algorithms from undirected to directed dependent failures in Section 5 and conclude in Section 6. Note that we can only sketch the proofs due to limitations of space. Refer to Warns et al.[19] for complete and fully detailed proofs.

2. The Model

Distributed systems. Our system model is based on the model of Fischer et al. [6]. We consider asynchronous distributed systems without bounds on message delay or relative process speeds. A system consists of a finite set of processes, $\Pi = \{p_1, \ldots, p_n\}$, which communicate by message-passing over reliable point-to-point channels. Each pair of processes is connected by such a channel.

A process is modelled as a sequential, deterministic automaton. The global state of the distributed system is a tuple of n states, one state for each process in Π. A global state is an initial global state if it only consists of initial states of the processes. A *run* r of the distributed system is a finite or infinite sequence $s_1, s_2, \ldots$ of global states where s_1 is an initial global state and where s_{i+1} results from s_i by executing a state transition on a single process and where the selection of the processes satisfies some fairness assumption. An *algorithm* A is a set of n sequential, deterministic automata, one for each process respectively. We denote the set of all runs that the algorithm can exhibit by ρ. In particular, we assume that ρ is not empty.

We define a *property* as a set of runs. A *problem* P consists of a set of properties that an algorithm should preserve. More precisely, a problem is the intersection of the sets of runs of its defining properties. An algorithm solves a problem P iff every run of the algorithm is a run of the problem P, that is, $\rho \subseteq P$.

Failure models. As we consider fault-tolerant systems in this paper, we need to specify the way in which a system may fail. This is done by a failure model. Intuitively, a *failure model* F is a set of properties that characterise all

IEEE COMPUTER SOCIETY

failures that may occur. Inspired by the *structure function model* of Echtle [5], we distinguish between functional and structural failure models.

A *functional failure model* describes the failure modes of single processes. For example, Cristian et al. [4] present a hierarchy of failure modes ranging from crash failures (i.e., prematurely halting) to Byzantine failures (i.e., arbitrary behaviour). Roughly speaking, a functional failure model transforms the automata of processes to allow "more" behaviour. Formally, a functional failure model may add states and transitions to an algorithm and may weaken the fairness assumption. For example, a bit flip of a register value can be modelled as an additional state transition from the original state into the corrupted state, and a crash failure can be modelled by weakening the fairness assumption such that a process is allowed to remain in a state even though there are outgoing transitions from that state. In general, more runs may occur under a functional failure model than in a failure-free system. The notion of functional failure models corresponds to *local failure assumptions* of Nordahl [17].

The focus of this paper is on structural failure models. A *structural failure model* restricts the failure modes of the functional failure model to a subset of processes. That is, it describes which processes may simultaneously fail in runs of a system. Therefore, it models the stochastic dependencies of failures. A structural failure model restricts the set of runs that can occur under a functional failure model. For example, if the functional model specifies that processes only fail by crashing, the structural failure model specifies that only a single process may fail, and one process has already failed, all other processes in the system need to be scheduled infinitely often. This notion corresponds to *global failure assumptions* of Nordahl [17]. A run of an algorithm running under a specific functional and structural failure model can be defined analogously to the failure-free definition above. In the following, we implicitly assume that an algorithm operates under some failure model.

We define structural failure models as follows: A process may either be correct (i.e., it behaves like given by the algorithm's automaton) or faulty (i.e., it shows a certain failure behaviour of the functional failure model) in a given run. A structural failure model characterises the set of all processes that can jointly fail in any run of the system. Formally, a structural failure model is defined by a non-empty subset of $\mathcal{P}(\Pi)$ containing sets of processes that may simultaneously fail in a run. However, structural failure models can be expressed in different ways as we will see later. We classify structural failure models into failure model classes. A structural failure model class $\mathcal{F}$ is a set of structural failure models. We say that a structural failure model F belongs to the class $\mathcal{F}$ iff $F \in \mathcal{F}$.

In this paper, we consider crash failures in the examples that illustrate our results. However, all our basic arguments only depend on the semantics of the structural failure model. Therefore, the main results on the relationships of different structural failure models are not specific to the crash failure mode, but apply to other modes such as Byzantine failures as well. We will give matching arguments for Byzantine failures respectively.

Comparing failure models. As we want to compare different classes of structural failure models, we define relations on these classes with respect to their expressiveness. We say that *a class of structural failure models $\mathcal{F}$ is weaker than a class $\hat{\mathcal{F}}$* if and only if everything that can be expressed by $\mathcal{F}$ can be expressed by $\hat{\mathcal{F}}$. If $\mathcal{F}$ is weaker than $\hat{\mathcal{F}}$, we write $\mathcal{F} \preceq \hat{\mathcal{F}}$. Formally,

$$\mathcal{F} \preceq \hat{\mathcal{F}} :\Leftrightarrow \mathcal{F} \subseteq \hat{\mathcal{F}}.$$

Intuitively, $\hat{\mathcal{F}}$ must describe at least as much failure behaviour as $\mathcal{F}$ does. If $\mathcal{F} \preceq \hat{\mathcal{F}}$ and $\hat{\mathcal{F}} \preceq \mathcal{F}$, we write $\mathcal{F} \cong \hat{\mathcal{F}}$ and say that $\mathcal{F}$ and $\hat{\mathcal{F}}$ are *equivalent*. If $\mathcal{F} \preceq \hat{\mathcal{F}}$ and there is a failure model F such that F is in $\hat{\mathcal{F}}$, but not in $\mathcal{F}$, we say that *$\mathcal{F}$ is strictly weaker than $\hat{\mathcal{F}}$* and write $\mathcal{F} \prec \hat{\mathcal{F}}$. Formally,

$$\mathcal{F} \prec \hat{\mathcal{F}} :\Leftrightarrow \mathcal{F} \subset \hat{\mathcal{F}}.$$

Note that each of these relations is transitive.

Consensus. The problem of *Consensus* is a fundamental agreement abstraction in fault-tolerant computing. Each process proposes a value and all processes must reach a common decision on a single value. More precisely, the Consensus problem is specified by three properties [3, 6]:

Validity Each value decided is a value proposed.

Agreement No two correct processes decide differently.

Termination Every correct processes eventually decides.

We will use Consensus as a benchmark for failure model classes, as described below.

Failure detectors. In their seminal paper, Fischer et al. [6] show that the problem of Consensus cannot be solved deterministically in the presence of even a single crash failure in an asynchronous system. To still be able to reason about Consensus in our examples, we augment our system model with failure detectors that are used by processes to detect failures of other processes [3]. Informally, failure detectors are components that are attached to each process. A process may query its failure detector, which indicates the set of processes that the detector suspects to have failed. Chandra and Toueg [3] introduce different classes of failure detectors. In this paper, we use the class of eventually strong failure detectors, denoted $\Diamond\mathcal{S}$.

IEEE COMPUTER SOCIETY

Tolerance bounds. The system and the failure model determine the tolerance bounds for a problem in fault-tolerant computing. A *tolerance bound* is a predicate on which sets of simultaneously failed processes can be tolerated by an algorithm that solves the problem. The structural failure model determines how to express these tolerance bounds. We will use Consensus as a benchmark problem to compare different failure model classes. Intuitively, a failure model class $\hat{\mathcal{F}}$ is more useful than a failure model class $\mathcal{F}$ iff $\hat{\mathcal{F}}$ allows better tolerance bounds for Consensus. Note that this implies that $\mathcal{F} \prec \hat{\mathcal{F}}$.

3. Relations among Previous Structural Failure Model Classes

The focus of this paper is on structural failure models that are suitable for fault-tolerant computing. For fault-tolerant computing, simplified models are employed that hide some aspects of failures to ease the development and evaluation of algorithms. Such models usually omit the actual probability values and provide no temporal information about failures. First, we present the class of commonly used threshold models that covers stochastically independent failures. Then, we present the general adversary structures, the fail-prone systems, and the core and survivor sets that cover undirected dependent failures. That is, failures may be correlated such that the failure of a process increases the failure probability of another process. However, this dependency holds the other way around resulting in an undirected correlation.

For every failure model class, we first state how it is expressed and then define the formal semantics of an instance of this class. Note that we define all classes in terms of our system model to ease the comparison of the models. The definitions are equivalent to the original ones in the literature.

Threshold Models. Threshold models allow to model t-out-of-n assumptions expressing that at most t out of a set of n processes simultaneously fail in a run. For example, if a system consists of five processes and at most two of them may fail simultaneously, this can modelled by the set of all sets of processes that contain two or less of the overall five processes.

Definition 1 (Threshold Models) *A failure model F is a* threshold model *if and only if it is expressed by a threshold $t \in \mathbb{N}$ and $\forall f \in \mathcal{P}(\Pi) : |f| \leq t \Rightarrow f \in F$.*

The class of threshold models is based on the assumption that it may happen in a run that any subset of $\mathcal{P}(\Pi)$ with i processes, $0 \leq i \leq t$, may fail simultaneously. Therefore, it only covers stochastically independent failures. Tolerance bounds for the threshold model are often expressed in the form of simple inequalities, $k \cdot t < n, k \in \mathbb{N}$, expressing that the number of processes must be greater than k times the maximal number of failed processes t. For example, if a system consists of n processes and the tolerance bound is that the majority of processes must be correct, then this can be stated by $2t < n$.

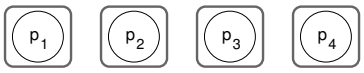

Figure 1. Illustration of threshold models

Example 3.1 Consider a system consisting of a set of processes $\Pi = \{p_1, \ldots, p_4\}$, which fail by crashing only, and a threshold such that *at most* 1 *of* 4 *processes fail*. As illustrated in Fig. 1, the threshold expresses a failure model $F = \{\{p_1\}, \{p_2\}, \{p_3\}, \{p_4\}, \{\}\}$. Within the figure, potential failures are illustrated by rounded rectangles. Any process may fail on its own, but two or more processes do not fail simultaneously. Assume that we intend to solve deterministic Consensus with an $\Diamond\mathcal{S}$ failure detector. The upper tolerance bound for such a system is $2t < n$ [3]. Therefore, our system with 4 processes can tolerate up to $t = 1$ failures. As the threshold satisfies the tolerance bound, the algorithm given by Chandra and Toueg [3] can be used to solve Consensus in this example. Note that the tolerance level is strict for this system, that is, no second failure can be tolerated.

General Adversary Structures. Originally, Hirt and Maurer [8] introduce the general adversary structures to specify potentially misbehaving players for secure multi-party computation. Subsequently, the models were applied to fault tolerance problems as well [1, 7]. They allow to strictly generalise results obtained with threshold models by enabling the specification of undirected dependent failures.

Definition 2 (General Adversary Structures) *A set of processes $A \subseteq \Pi$ is an* adversary class *for a failure model F iff there exists $f \in F$ such that $f = A$. A failure model is a* general adversary structure *iff it is expressed as a monotone set $\mathcal{Z} \subseteq \mathcal{P}(\Pi)$ of all adversary classes for F, where monotone means $\forall A \in \mathcal{Z}, \forall \hat{A} \subseteq A : \hat{A} \in \mathcal{Z}$.*

The class of general adversary structures is based on the assumption that if a certain set of processes may fail, then it is also possible that only its subsets fail. For example, if $\{p_1, p_2\}$ is an adversary class, it must also be possible that in some runs only p_1, only p_2, or no process at all fails.

Tolerance bounds for general adversary structures are expressed in terms of the predicate $Q^{(k)}$ [7, 8]. A general adversary structure $\mathcal{Z}$ satisfies the predicate $Q^{(k)}$ if and only

IEEE COMPUTER SOCIETY

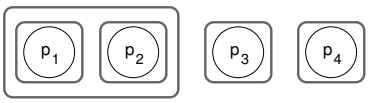

Figure 2. Illustration of adversary structures

if

$$\forall A_1, \ldots, A_k \in \mathcal{Z} : A_1 \cup \ldots \cup A_k \neq \Pi.$$

For example, Fitzi and Maurer [7] show that Agreement can be solved for a general adversary structure despite Byzantine failures in a synchronous system iff the structure satisfies $Q^{(3)}$.

The $Q^{(k)}$ predicate is a generalisation of the $k \cdot t < n$ inequality of threshold models. If at most t processes fail and n is more than k times greater than t, then no k sets of failed processes cover the whole process set. However, if the $Q^{(k)}$ predicate is satisfied by a general adversary structure, more than t processes may fail as we now explain.

Example 3.2 Consider a system consisting of a set of processes $\Pi = \{p_1, \ldots, p_4\}$, which fail by crashing only, and a general adversary structure $\mathcal{Z} = \{\{p_1, p_2\}, \{p_1\}, \{p_2\}, \{p_3\}, \{p_4\}, \{\emptyset\}\}$ satisfying $Q^{(2)}$ as illustrated in Fig. 2. Any process may fail on its own. The processes p_1 and p_2 may fail simultaneously, but no other combination of processes may do so. This example illustrates that the class of general adversary structures cannot be weaker than the class of threshold models as there is no threshold model to model this specific failure assumption. Since $\mathcal{Z}$ satisfies $Q^{(2)}$, it is possible to solve Consensus using an $\Diamond\mathcal{S}$ failure detector under this failure model (see below). This shows that general adversary structures allow to generalise results obtained with threshold models. Only one arbitrary failure can be tolerated for the same system using threshold models, but one arbitrary or two specific failures can be tolerated using general adversary structures.

Mostéfaoui and Raynal [16] present an algorithm that solves Consensus using failure detectors under threshold models. A major advantage of this algorithm is its flexibility achieved by formulating important predicates in terms of specific quorum sets. For example, the algorithm can be used with failure detectors of different classes by using appropriate quorum sets. Hurfin et al. [9] generalise the algorithm to define a versatile family of Consensus algorithms under the threshold model substantiating the advantages of the algorithm. It is relatively easy to adapt the algorithm to work with general adversary structures: The algorithm solves Consensus with an $\Diamond\mathcal{S}$ failure detector and a general adversary structure satisfying $Q^{(2)}$ if the quorum set only consists of all complements of maximal adversary classes [19]. We are unaware of other algorithms that solve asynchronous Consensus using failure detectors and general adversary structures assuming crash failures.

Fail-Prone Systems. Malkhi and Reiter [15] introduce fail-prone systems in the context of quorum systems for data repositories that are able to tolerate Byzantine failures. Fail-prone systems are closely related to general adversary structures as a fail-prone system is simply expressed as the set of all maximal adversary classes. The semantics of the failure model, that is, which sets of processes are allowed to fail simultaneously, are a general adversary structure.

Definition 3 (Fail-prone System) *An adversary class A for a failure model F is* maximal *for F if and only if there is no f in F such that A for F is a proper subset of f. A failure model F is a fail-prone system if and only if F is monotone and it is expressed as a set $\mathcal{B} \subseteq \mathcal{P}(\Pi)$ of all maximal adversary classes for F.*

It is obvious that general adversary structures can be obtained from fail-prone systems and vice versa. Therefore, both have the same expressive power and are equivalent. Furthermore, an algorithm for general adversary structures can be easily transformed to be used with fail-prone systems and the other way around.

Core and Survivor Sets. Junqueira and Marzullo [10, 11] introduce the notions of core and survivor sets and use this class of failure models to solve Consensus. A core is a set of processes that contains at least one correct process. Moreover, the set is *minimal*, that is, removing some process implies that there is a run $r \in \rho$ such that the core contains no correct process for r.

Definition 4 (Core and Core Set) *A set of processes $C \subseteq \Pi$ is a* core *for a failure model F iff $(\forall f \in F, \exists p \in C : p \notin f) \wedge (\forall p \in C, \exists f \in F, \forall \hat{p} \in C \setminus \{p\} : \hat{p} \in f)$. A failure model F is a* core set *iff it is expressed as a set $\mathcal{C} \subseteq \mathcal{P}(\Pi)$ of all cores for F.*

For example, consider a system for which each process of a set of processes $\Pi = \{p_1, \ldots, p_4\}$ may fail on its own and only p_1 and p_2 may fail simultaneously. Then, $C = \{p_1, p_3\}$ is a core, because p_1 or p_3 is correct for each run and there are runs such that either p_1 or p_3 fails.

Additionally, Junqueira and Marzullo [11] introduce survivors and survivor sets. A survivor is a set of processes that has a common process with each core. Moreover, the set is *minimal*, that is, removing any process implies that there is a core, which has no common process with the survivor anymore. Junqueira et al. [12] give a bijective mapping between equivalent core and survivor sets showing that the classes of core and survivor sets are equivalent.

IEEE COMPUTER SOCIETY

Definition 5 (Survivor and Survivor Set) *Given a particular failure model F expressed by a core set $\mathcal{C}$, a set of processes $S \subseteq \Pi$ is a* survivor *for F iff* $(\forall C \in \mathcal{C} : S \cap C \neq \emptyset) \wedge (\forall p \in S, \exists C \in \mathcal{C} : [p \in C] \wedge [(S \setminus \{p\}) \cap C = \emptyset])$. *A failure model F is a* survivor set *iff it is expressed as a set $\mathcal{S} \subseteq \mathcal{P}(\Pi)$ of all survivors for F.*

We slightly changed the terms for core and survivor sets compared to the original definitions. Junqueira and Marzullo use the term *survivor set* and *set of survivor sets* for what we call survivor and survivor set respectively.

The following lemma illustrates a key property of survivor sets $\mathcal{S}$.

Lemma 3.1 *For a survivor set $\mathcal{S}$ expressing a failure model F, for each set of processes $f \in F$, there exists a survivor $S \in \mathcal{S}$ that only contains processes that are not in f.*

PROOF SKETCH: We fix a set of processes f in F. It suffices to show that there is a survivor for F that only contains processes that are not in f. In particular, there is a set $\hat{S} \subseteq \Pi$ of processes that are not in f. Furthermore, it can be shown that there is a subset S of $\hat{S}$ that is a survivor for F[19]. By the definition of survivor sets, S is included in $\mathcal{S}$. As $\hat{S}$ only contains processes that are not in f, this is true for S as well, because S is a subset of $\hat{S}$.

Another key property of $\mathcal{S}$ expressing F is that all processes in the complement of S may fail simultaneously.

Lemma 3.2 *For each survivor S in the survivor set $\mathcal{S}$ expressing a failure model F, there is a set of processes f in F such that all processes in the complement of S are in f.*

PROOF SKETCH: We assume the opposite and obtain a contradiction. We can construct a set $\hat{C} = \cup_{f \in F}\{p : p \in \Pi \setminus S, p \notin f\}$ such that $S \cap \hat{C} = \emptyset$ and $\forall f \in F, \exists p \in \hat{C} : p \notin f$. In particular, it can be shown that $\hat{C}$ has a subset C that is a core for F[19]. As C is a subset of $\hat{C}$, the intersection of S and C is empty, that is, $S \cap C = \emptyset$. By the definition of survivors, S is not a survivor for F then.

Junqueira and Marzullo [10] show that the core and survivor sets allow to generalise results obtained with the threshold model. In particular, they give tolerance bounds for asynchronous Consensus with failure detectors. They reason that the *Crash Partition* property is necessary and sufficient to solve Consensus with a failure detector equivalent to an $\Diamond\mathcal{S}$ failure detector. The property is satisfied iff every partition (A, B) of a set of processes Π is such that either A or B contain a core. Furthermore, they show that the *Byzantine Partition* condition is necessary and sufficient to solve Consensus in a synchronous system. The property is satisfied iff every partition (A, B, C) of a set of processes Π is such that either A, B, or C contain a core.

Example 3.3 Consider a system with a set of processes $\Pi = \{p_1, \ldots, p_4\}$, which fail by crashing only, and a core set $\mathcal{C} = \{\{p_1, p_3\}, \{p_1, p_4\}, \{p_2, p_3\}, \{p_2, p_4\}, \{p_3, p_4\}\}$. Using the mapping of Junqueira et al. [12], the equivalent survivor set is $\mathcal{S} = \{\{p_3, p_4\}, \{p_1, p_2, p_3\}, \{p_1, p_2, p_4\}\}$. As a survivor set models the processes that work correctly at least, either the processes in $\{p_3, p_4\}$, in $\{p_1, p_2, p_3\}$, or in $\{p_1, p_2, p_4\}$ work correctly at least. Taking the complements, either p_1 and p_2, p_3, or p_4 simultaneously fail at most. Obviously, this models the same failure assumptions as the general adversary structure of Example 3.2, which is illustrated in Fig. 2. As $\mathcal{C}$ satisfies the Crash Partition property, the algorithm of Junqueira and Marzullo [10] solves asynchronous Consensus with failure detectors in this case.

Relationships among Models. Different failure model classes can be compared with respect to their expressiveness. As the examples given above indicate, the class of threshold models is strictly weaker than the class of general adversary structures, fail-prone systems, and core and survivor sets. We now argue that the classes of general adversary structures, fail-prone systems, and core and survivor sets are equivalent.

Theorem 1 *The class of threshold models is strictly weaker than the class of general adversary structures, fail-prone systems, and core and survivor sets. The latter classes are equivalent.*

PROOF SKETCH: Fitzi and Maurer [7] show that the class of threshold models is strictly weaker than the class of general adversary structures. The equivalence between the classes of fail-prone systems and general adversary structures is obvious. In order to prove the equivalence with the class of core and survivor sets, consider mappings $cs2gas$ and $gas2cs$ from survivor sets to general adversary structures and vice versa. In particular, we define $cs2gas$ as follows: for a survivor set $\mathcal{S}$, $cs2gas$ returns a general adversary structure $\mathcal{Z}$ using the complements of all survivors in $\mathcal{S}$ as (maximal) adversary classes and adding subsets thereof until $\mathcal{Z}$ is monotone. For the opposite direction, we define $gas2cs$ as follows: for a general adversary structure $\mathcal{Z}$, $gas2cs$ returns a survivor set $\mathcal{S}$ using the complements of maximal adversary classes as survivors. To prove the equivalence of both structural failure model classes, it suffices to show that both mappings map between models expressing the same failure model F. Essentially, this proof can be derived from the two key observations of Lemma 3.1 and Lemma 3.2. However, due to limitations of space, we cannot give the proof in further detail. Refer to Warns et al. [19] for the full details of the proof.

IEEE COMPUTER SOCIETY

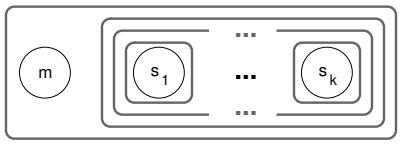

Figure 3. Illustration of the DiDep class

4. DiDep

In this section, we introduce the class of DiDep models that is more expressive than previous classes as it allows to model directed dependent failures. Furthermore, DiDep models allow to generalise results obtained with previous models. Similar to previous models, they abstract from temporal dependencies and the probabilistic nature of failures, that is, they provide no information on temporal order and the actual probability values. DiDep models are closely related to general adversary structures as they are sets of adversary classes as well. The main difference is that monotonicity is not required for a DiDep model.

Definition 6 (DiDep Model) *A failure model F is a* DiDep model *iff it is expressed as a set $\mathcal{D} \subseteq \mathcal{P}(\Pi)$ of all adversary classes of F.*

It is obvious that the class of general adversary structures is weaker than the class of DiDep models, because each general adversary structure is a DiDep model as well. As a general adversary structure $\mathcal{Z}$ is monotone, it provides the following information when a process p fails: Specific other processes *may* fail as well if they are in an adversary class of $\mathcal{Z}$ together with p. Analogously, specific other processes do not fail together with p if there is no adversary class in $\mathcal{Z}$ that contains them together with p. In contrast, a DiDep model $\mathcal{D}$ can provide more information. In particular, if there is a process $\hat{p}$ such that $\hat{p}$ is in each adversary class of $\mathcal{D}$ that contains p as well, $\hat{p}$ *must* fail jointly with p.

Example 4.1 Consider a set of processes $\Pi = \{m, s_1, \ldots, s_k\}$, which fail by crashing only, and a DiDep model $\mathcal{D} = \{\{m, s_1, \ldots, s_k\}\} \cup \mathcal{P}(\{s_1, \ldots, s_k\})$ as illustrated in Fig. 3. Any subset of $\{s_1, \ldots, s_k\}$ may fail on their own, but all processes fail if m fails, because all processes are in the only adversary class that contains process m.

The example shows that the class of DiDep models cannot be weaker than the class of general adversary structures, because there is no general adversary structure for this specific failure assumption. As the class of general adversary structures is weaker than the DiDep class, all classes of undirected dependent failure models are strictly weaker than the DiDep class by Theorem 1.

Algorithm 1: MasterConsensus(v_p).

```
   Input: Proposed value v_p
   Output: Decided value v
 1 if p = m then
 2     forall s ∈ {s_1, ..., s_k} do
 3         send DECIDE(v_p) to s
 4     end
 5     return v
 6 else
 7     wait until DECIDE(v) received from m
 8     return v
 9 end
```

Furthermore, the example illustrates the reason for the increase in expressiveness is that DiDep allows to specify directed dependent failures. If process m fails, all other processes fail as well, but if a process $s_i, 1 \leq i \leq k$, fails, m may not fail. Compare this to a specification by a general adversary structure $\mathcal{Z}$. As all processes may jointly fail, Π must be in $\mathcal{Z}$. Hence, all possible subsets of Π are in $\mathcal{Z}$ as well, because $\mathcal{Z}$ is monotone. This introduces the implicit assumption that some $s_i, 1 \leq i \leq k$, can be correct even if m fails, although there is no such run.

Apart from being more expressive, the DiDep class also allows to increase the resilience of solutions to Consensus. Algorithm 1 solves asynchronous Consensus deterministically even *without* failure detectors for the failure model given in the example. Therefore, it opens another condition under which the impossibility results of Fischer et al. [6] do not hold. In the algorithm, the master sends its proposal value to all slave processes and decides. The slave processes wait for this message and decide if they receive it. The critical property of Consensus is Termination (Agreement and Validity are easy to verify). The algorithm looks like slave processes may wait infinitely for the message from the master in the case of a master crash. Because of the DiDep failure model however, the slaves jointly crash if the server crashes. So if m is correct, all correct processes eventually receive the decision value from m and decide on this value. Otherwise, no process is correct and nobody decides. By similar arguments, it can be shown that the algorithm even solves Consensus in the presence of Byzantine failures if processes can correctly determine the sender of a received message.

5. Solving Consensus using DiDep

The class of DiDep models allows to express failure behaviour in more detail and, therefore, it has more expressive power. We now show that this does not significantly penalise simplicity, but allows to generalise results obtained

IEEE COMPUTER SOCIETY

Algorithm 2: extract_undep($\mathcal{D}$).

Input: DiDep model $\mathcal{D}$
Output: Set of processes $\mathcal{U}$ whose failures are undirected dependent

```
1 U ← Π
2 while ∃p, p̂ ∈ U : [p ≠ p̂] ∧ [∀A ∈ D : p̂ ∈ A ⇒ p ∈ A] do
3     U ← U \ {p}
4 end
5 return U
```

Algorithm 3: extract_structure($\mathcal{D}$).

Input: DiDep model $\mathcal{D}$
Output: General Adversary Structure $\mathcal{Z}_\mathcal{U}$

```
1 U ← extract_undep(D)
2 Z_U ← {}
3 forall A ∈ D do
4     Z_U ← Z_U ∪ {A|_U}
5 end
6 return Z_U
```

Algorithm 4: ConsensusDiDep(v_p).

Input: Proposed value v_p
Output: Decided value v

```
1 U ← extract_undep(D)
2 if p ∈ U then
3     v ← ConsensusAlgorithm(v_p)
4     forall p̂ ∈ Π \ U do send DECIDE(v) to p̂
5 else
6     wait until DECIDE(v) received
7 end
8 return v
```

with previous models. We give a Consensus algorithm for DiDep models that has relaxed tolerance bounds compared to solutions using previous models. We apply a transformational approach that uses an arbitrary Consensus algorithm for general adversary structures (or another model for undirected dependent failures) instead of developing a new one from scratch. In particular, we show how to identify a minimal set of processes such that all processes depend on a process in this set. Within the algorithm for DiDep models, these processes solve Consensus with an algorithm using general adversary structures and propagate the decided value to the remaining processes.

Dependency relation. The specification of directed dependent failures introduces a dependency relation among processes. We say that a process p depends on a process $\hat{p}$ under a failure model F if p fails when $\hat{p}$ fails, that is, $\forall f \in F : \hat{p} \in f \Rightarrow p \in f$.

Extracting independent processes. In Example 4.1, each process depends on itself, and each process $s_i, 1 \leq i \leq k$, depends on m. In general, there is a subset of processes such that every process depends on a process within this subset (*dependency property*). Algorithm 2 extracts such a set $\mathcal{U}$ of processes for an arbitrary DiDep model.

Lemma 5.1 *Let $\mathcal{D}$ be a DiDep model expressing a failure model F and let $\mathcal{U} = \text{extract\_undep}(\mathcal{D})$. Each process p depends on a process $\hat{p}$ in $\mathcal{U}$, that is, p fails if $\hat{p}$ fails. Formally, $\forall p \in \Pi, \exists \hat{p} \in \mathcal{U}, \forall f \in F : \hat{p} \in f \Rightarrow p \in f$.*

PROOF SKETCH: The algorithm initially starts with the whole process set Π for which the dependency property is obviously satisfied. As it only removes a process from $\mathcal{U}$ if this process depends on another process in $\mathcal{U}$ and the dependency relation is transitive, the dependency property is never violated. The algorithm terminates, because Π is finite, one process is removed each time the while loop is executed, and the while condition is false for the empty set.

Independent processes. The set that is returned by Algorithm 2 is minimal meaning that no process can be removed from $\mathcal{U}$ without violating the dependency property, because it removes as much processes as possible until termination. As $\mathcal{U}$ is minimal, no process in $\mathcal{U}$ depends on another process in $\mathcal{U}$. Therefore, the failure assumption on the processes in $\mathcal{U}$ can be modelled by a general adversary structure if the view is restricted to $\mathcal{U}$. Formalising such a restriction, we say that the set $A|_B$ is the set A restricted to B iff $A|_B = A \cap B$. In particular, Algorithm 3 extracts the general adversary structure $\mathcal{Z}_\mathcal{U}$ for a given DiDep model $\mathcal{D}$ with $\mathcal{U} = \text{extract\_undep}(\mathcal{D})$.

Lemma 5.2 *Let $\mathcal{D}$ be a DiDep model expressing F and let $\mathcal{U} = \text{extract\_undep}(\mathcal{D})$. The set $\mathcal{Z}_\mathcal{U} = \text{extract\_structure}(\mathcal{D})$ is the general adversary structure for $F|_\mathcal{U}$.*

PROOF SKETCH: The algorithm only adds adversary classes to $\mathcal{Z}_\mathcal{U}$ by considering all sets A in $\mathcal{D}$ restricted to $\mathcal{U}$. It does not overlook any adversary classes, because it considers all adversary classes in $\mathcal{D}$. The algorithm terminates, because extract_undep terminates and the number of adversary classes is finite.

With such a general adversary structure, Consensus can be solved for a DiDep model by using a Consensus algorithm for general adversary structures, for example, the algorithm given by Warns et al. [19]. This algorithm solves Consensus among the processes in $\mathcal{U} = \text{extract\_undep}(\mathcal{D})$ if $\mathcal{Z}_\mathcal{U} = \text{extract\_structure}(\mathcal{D})$ satisfies $Q^{(2)}$ with respect to $\mathcal{U}$ in the presence of crash failures.

The transformer. The processes in $\mathcal{U}$ play a special role in Algorithm 4 that solves Consensus for the DiDep model. If these processes are able to decide a value among themselves with an arbitrary Consensus algorithm, Consensus

IEEE COMPUTER SOCIETY

can be solved for the overall process set. If the processes in $\mathcal{U}$ decide a value, they propagate this value to the remaining processes, which wait for notification from a process in $\mathcal{U}$. In particular, if a process p that is not in $\mathcal{U}$ is correct, then there is a process in $\mathcal{U}$ that is correct as well, because p depends on a process in $\mathcal{U}$. Hence, p receives a notification from a process in $\mathcal{U}$. Note that the following theorem does not state any assumptions on failure detectors. However, the Consensus algorithm used within the algorithm may require such assumptions.

Theorem 2 *Algorithm 4 solves Consensus in the presence of crash failures in an asynchronous system for a DiDep model $\mathcal{D}$ if* ConsensusAlgorithm *solves Consensus for* $\mathcal{U} = \text{extract\_undep}(\mathcal{D})$.

PROOF SKETCH: Each value decided is a value proposed as it is returned by ConsensusAlgorithm, which is correct by assumption. No two correct processes decide differently, because ConsensusAlgorithm returns the same value for each process in $\mathcal{U}$. If a process is in $\mathcal{U}$, it eventually decides, because ConsensusAlgorithm terminates and there is no other point to block. If a process is not in $\mathcal{U}$, it eventually decides, because the process eventually receives a value from a correct process in $\mathcal{U}$.

Interestingly, the algorithm resembles some ideas of the Paxos algorithm given by Lamport [14]. For Paxos, three roles for processes are differentiated: *proposers*, *acceptors*, and *learners*. A proposer proposes values to decide and an acceptor decides on proposed values while coordinating with other acceptors. A learner is only informed about a decided value. Within our algorithm, all processes are proposers as each process proposes a value. However, the processes in $\Pi \setminus \mathcal{U}$ only propose values due to the Consensus problem specification: their proposed values do not influence the decision process. Only the processes in $\mathcal{U}$ are acceptors as only the processes in $\mathcal{U}$ participate in the decision process. Finally, all processes are learners as all learn about a decided value.

Theorem 2 illustrates that the DiDep model allows to generalise results obtained with previous models. If $\mathcal{D}$ is a DiDep model, the set $\mathcal{U}$ is a subset of the overall set of processes Π. ConsensusAlgorithm must solve Consensus among these processes only. The set $\mathcal{Z}_\mathcal{U}$ must satisfy $Q^{(2)}$ with respect to to $\mathcal{U}$ only. In particular, $Q^{(2)}$ may be violated with respect to Π as seen in Example 4.1.

Example 5.1 Consider a system that is a set of processes $\Pi = \{m_1, \ldots, m_4, s_1, \ldots, s_l\}$, which fail by crashing only, and a DiDep model $\mathcal{D} = \{\{m_1, m_2, s_1, \ldots, s_j\}, \{m_1, s_1, \ldots, s_i\}, \{m_2, s_{i+1}, \ldots, s_j\}, \{m_3, s_{j+1}, \ldots, s_k\}, \{m_4, s_{k+1}, \ldots, s_l\}\} \cup \mathcal{P}(\{s_1, \ldots, s_l\})$. Each combination of processes in $\{s_1, \ldots, s_l\}$ may fail jointly. Each process in $m_1, \ldots, m_4$ may fail without another $m_h, 1 \leq h \leq 4$, and m_1 and m_2 may fail jointly. If m_1 fails, all processes in $\{s_1, \ldots, s_i\}$ fail as well. If m_2 fails, all processes in $\{s_{i+1}, \ldots, s_j\}$ fail as well. If m_3 fails, all processes in $\{s_{j+1}, \ldots, s_k\}$ fail as well. If m_4 fails, all processes in $\{s_{k+1}, \ldots, s_l\}$ fail as well. Therefore, each process depends on a process in $\{m_1, \ldots, m_4\}$. Algorithm 2 returns the set $\mathcal{U} = \{m_1, \ldots, m_4\}$. Algorithm 3 returns the set $\mathcal{Z}_\mathcal{U} = \{\{m_1, m_2\}, \{m_1\}, \{m_2\}, \{m_3\}, \{m_4\}, \emptyset\}$. As $\mathcal{Z}_\mathcal{U}$ satisfies $Q^{(2)}$ with respect to $\mathcal{U}$, Consensus can be solved among the processes of $\mathcal{U}$ by an algorithm for general adversary structures. Therefore, Algorithm 4 solves Consensus for $\mathcal{D}$ if it uses such an algorithm.

It is relatively easy to extend Algorithm 4 to Byzantine failures. There is a process $\hat{p}$ in $\mathcal{U}$ for each process p not in $\mathcal{U}$ such that p depends on $\hat{p}$. If p only decides the value received from $\hat{p}$, then agreement is solved for Byzantine failures as well, because, by assumption, p fails if $\hat{p}$ fails.

6. Discussion and Conclusions

The application of fault-tolerant algorithms in new contexts such as intrusion-tolerant systems yields new challenges for their development and evaluation. In particular, it requires to adapt structural failure models, because the common assumption of stochastic independent failures may not be valid anymore. We have presented structural failure models that are suitable for fault-tolerant computing and allow to specify dependent failures. In particular, we have introduced the DiDep class that covers directed dependent failures.

From a theoretical point of view, the class of DiDep models can be regarded as the "most expressive" structural failure model class (omitting probability values and timing information), because its semantics are equivalent to the general form of a structural failure model as a set of subsets of Π that can jointly fail. Therefore, any failure model can trivially be expressed as a DiDep model. The major advantage of DiDep is that its expressiveness allows to model failure assumptions more accurately than using previous classes. In general, if a failure model does not express failure assumptions with an appropriate accuracy, the failure model is either too optimistic or too pessimistic. If the model is too optimistic, the model expresses less failure behaviour than occurring in reality such that the system suffers from low assumption coverage and fails even if proven to be correct. If the model is too pessimistic, the model expresses more failure behaviour than occurring in reality such that the systems is less resilient than possible: The system either requires an unnecessary amount of redundancy for a given number of failures to tolerate or tolerates less failures than possible for a given amount of redundancy. In sum-

IEEE COMPUTER SOCIETY

mary, DiDep as a more expressive model allows to improve the assumption coverage and the resilience of a system.

The class of DiDep models has another interesting theoretical property: a DiDep model is a combination of a *safety* and a *liveness* property on failures. On the one hand, a DiDep model states that no more failures than specified occur. On the other hand, it may state that certain specified failures will occur eventually. For example, a DiDep model $\mathcal{D}$ can express that a process p depends on another process $\hat{p}$ such that p eventually fails if $\hat{p}$ has failed. In contrast, threshold models and the undirected dependent failure models are only safety properties on failures. For example, the monotonicity of general adversary structures implies that each subset of an adversary class for a failure model F is an adversary class for F as well. Subsequently, a general adversary structure cannot state that a process must fail jointly with another process. Therefore, a general adversary structure is only a *safety* property on failures stating that no more failures occur than specified by its adversary classes.

For the moment, we only combined DiDep models being structural failure models with the functional failure model of crashes. In future work, we will combine DiDep models with other functional failure models and, in particular, transient failures such as the *crash-recovery* model. Under a crash-recovery model, processes are allowed to crash and, subsequently, recover during the execution of an algorithm. Currently, solutions using the crash-recovery model are developed under a variant of the threshold model assuming that, eventually, at most t of n processes do not work correctly. The major difference to the classic threshold model is that the threshold must hold eventually only. This idea will be applied to other structural failure models as a part of our future work. Furthermore, we will investigate hybrid failure models addressing situations in which, for example, both crash and Byzantine failures may occur. Finally, we will look into the efficiency of algorithms using DiDep models illustrating that DiDep also allows to reduce the number of rounds and messages of solutions.

References

[1] C. Cachin. Distributing trust on the internet. In *Proc. 2001 Int'l Conf. Dependable Systems and Networks (DSN '01)*, pages 183–192. IEEE Computer Society Press, 2001.

[2] C. C. Center. CERT advisory CA-1996-26 denial-of-service attack via ping. Internet: `http://www.cert.org/advisories/CA-1996-26.html`, Dec. 1996.

[3] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, Mar. 1996.

[4] F. Cristian, H. Aghili, R. Strong, and D. Dolev. Atomic broadcast: From simple message diffusion to Byzantine agreement. *Information and Computation*, 118(1):158–179, Apr. 1995.

[5] K. Echtle. *Fehlertoleranzverfahren*. Springer, 1990.

[6] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, Apr. 1985.

[7] M. Fitzi and U. M. Maurer. Efficient byzantine agreement secure against general adversaries. In S. Kutten, editor, *Proc. 12th Int'l Symp. Distributed Computing (DISC '98)*, volume 1499 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 1998.

[8] M. Hirt and U. Maurer. Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract). In *Proc. 16th ACM Symp. Principles of Distributed Computing (PODC '97)*, pages 25–34. ACM Press, 1997.

[9] M. Hurfin, A. Mostéfaoui, and M. Raynal. A versatile family of consensus protocols based on Chandra-Toueg's unreliable failure detectors. *IEEE Trans. Computers*, 51(4):395–408, 2002.

[10] F. P. Junqueira and K. Marzullo. Designing algorithms for dependent process failures. In A. Schiper, A. A. Shvartsman, H. Weatherspoon, and B. Y. Zhao, editors, *Future Directions in Distributed Computing*, volume 2584 of *Lecture Notes in Computer Science*, pages 24–28. Springer, 2003.

[11] F. P. Junqueira and K. Marzullo. Synchronous consensus for dependent process failures. In *Proc. 23rd Int'l Conf. Distributed Computing Systems (ICDCS '03)*, pages 274–283. IEEE Computer Society Press, 2003.

[12] F. P. Junqueira, K. Marzullo, and G. M. Voelker. Coping with dependent process failures. Technical Report CS2002-0723, University of California, San Diego, Dec. 2001.

[13] I. Keidar and K. Marzullo. The need for realistic failure models in protocol design. In *Proc. 4th Information Survivability Workshop (ISW 2001/2002)*, Mar. 2002.

[14] L. Lamport. The part-time parliament. *ACM Trans. Computer Systems*, 16(2):133–169, 1998.

[15] D. Malkhi and M. Reiter. Byzantine quorum systems. In *Proc. 29th ACM Symp. Theory of Computing (STOC '97)*, pages 569–578. ACM Press, 1997.

[16] A. Mostéfaoui and M. Raynal. Solving consensus using chandra-toueg's unreliable failure detectors: A general quorum-based approach. In *Proc. 13th Int'l Symp. Distributed Computing (DISC '99)*, pages 49–63. Springer, 1999.

[17] J. Nordahl. *Specification and Design of Dependable Communicating Systems*. PhD thesis, Technical University of Denmark, Department of Computer Science, 1992.

[18] P. Veríssimo, N. F. Neves, and M. P. Correia. Intrusion-tolerant architectures: Concepts and design. In R. de Lemos, C. Gacek, and A. Romanovsky, editors, *Architecting Dependable Systems*, volume 2677 of *Lecture Notes in Computer Science*. Springer, 2003.

[19] T. Warns, F. C. Freiling, and W. Hasselbring. Solving consensus using structural failure models. Technical Report 03-06, University of Oldenburg, Department of Computing Science, Apr. 2006.

[20] J. H. Wensley, L. Lamport, J. Goldberg, M. W. Green, K. N. Levitt, P. M. Melliar-Smith, R. E. Shostak, and C. B. Weinstock. SIFT: Design and analysis of a fault-tolerant computer for aircraft control. *Proc. IEEE*, 66(10):1240–1255, Oct. 1978.

0-7695-2677-2/06 $20.00 © 2006 IEEE

IEEE COMPUTER SOCIETY