How Resilient is your computer system?

William Excoffon, Jean-Charles Fabre, Michaël Lauer

▶ **To cite this version:**

William Excoffon, Jean-Charles Fabre, Michaël Lauer. How Resilient is your computer system?. ERTS 2018,9th European Congress Embedded Real Time Software and systems, Jan 2018, Toulouse, France. hal-01708220

HAL Id: hal-01708220
https://hal.science/hal-01708220

Submitted on 13 Feb 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

How Resilient is your computer system?

W. Excoffon[1], J.-C.Fabre[1], M. Lauer[2]

CNRS-LAAS, Ave du Colonel Roche, F-31400 Toulouse, France
Univ de Toulouse, [1]INP, [2]UPS, LAAS, F-31400 Toulouse, France

***Abstract*—A system that remains dependable when facing changes (new threats, failures, updates) is called resilient. The fast evolution of systems, including embedded systems, implies modifications of applications and system configuration, in particular at software level. Such changes may have an impact on the dependability of the system. A system is resilient when such changes do not invalidate its dependability mechanisms, said in a different way, current dependability mechanisms remain appropriate despite changes.**

In this paper we introduce some measures to quantify the capability of a system to remain dependable despite changes, i.e. how resilient it is!

I. INTRODUCTION

Today systems evolve very fast during their service life for several reasons, including additional features requested by users. This is true for conventional systems offering services to users like data servers, telephony, messaging, banking, and numerous remote services accessible via the Internet. Any user is today equipped with one or more devices and customer of many services. Embedded systems are not outside of this trend. They become open! This is true for the automotive industry with the development of connected cars. For dependable systems, the challenge is greater, as evolution must not impair dependability attributes.

The persistence of system dependability when facing changes is called resilience [1].

We use here the definition of resilience given above and proposed by JC. Laprie and colleagues in the *ReSIST*[1] *Network Of Excellence*. The difference between resilience and dependability is very important. Resilient computing adds a new dimension to fault tolerant computing by including change events, in particular unexpected changes related to application updates, new fault tolerance requirements, system configuration changes, etc. In this respect, fault tolerance strategies installed at one time can be invalidated by a change later on. Although a system may be developed with adequate dependability mechanisms at a given point in time, some evolution may have unexpected side effects. Evolutions encompasses, but are not limited to, corrective maintenance, updates and upgrades.

The word "Resilience" is often used in Cloud computing [2] and networking as a synonymous to fault tolerance that is not consistent with our definition. However, the metrics used in these fields (data losses, CPU efficiency…) are analog to Fault Tolerance measures. This work proposes an approach to measure the impact of unexpected change events on the persistence of dependability. The proposed measures are complementary to standard fault tolerance measures.

Dependability relies at runtime on fault tolerance mechanisms attached to the application [3]. A challenge of resilient computing is to maintain the adequacy between any application and its attached FTMs (*Fault Tolerance Mechanism*) during the operational life of the system despite change events.

In this paper, we propose some measures to estimate the resilience of a system. These measure do not compete with conventional dependability measures, they are complementary. We finally draw some lessons learnt for the development of resilient systems.

II. RESILIENT COMPUTING & PROBLEM STATEMENT

Resilient computing raises several challenges in various dimensions (*evolvability*, *assessability*, *usability*, *diversity*) and call for resilience scaling technologies. Examples of challenges up are the following, as identified in ReSIST:

- for *evolvability*, to be able to adapt to changing environments and threats,
- for *assessability*, to move from off-line, pre-deployment assessment to continuous automated and operational assessment,
- for *usability*, to reconcile the conflicting roles of humans as contributors to resilience and threats that resilience must tolerate,
- for *diversity*, to take advantage of diversity in order to prevent some vulnerabilities from becoming single points of failure.

In the work reported in this paper we address the first of these challenges: *evolvability* and its impact on dependability. The aim is to find measures to assess the resilience of a system when changes are performed. Such changes refer to both application and dependability components. They can be mandatory to maintain the system consistent with the specifications, both functional and non-functional. Such changes follow a random process as they can occur when a bug is discovered. This type of change is related corrective maintenance. Other changes can be voluntary in the sense that they are performed to improve the system, e.g. updates to improve for instance the implementation of given application, or upgrades when some additional features are added to the system, for commercial reasons for instance. All these changes may have an impact on the dependability of the system.

The key point here relates to the importance of the assumptions regarding dependability mechanisms, in

[1] ReSIST NoE, Resilience for Survivability in IST, http://www.resist-noe.org/.

particular fault tolerance mechanisms. Such assumptions refer to the considered fault model, behavioral and structural characteristics of the application, resources of the system, etc. The coverage of these assumptions is a very important figure to estimate the real efficiency of a fault tolerance mechanism, provided the mechanism is correctly implemented. An FTM remains useful and efficient while changes do not affect its assumptions. When changes do not affect the FTM assumptions, then the system is considered resilient. When changes affect one of its assumptions, then the FT mechanism becomes invalid and by the way useless. It must be adapted i) to comply with the assumptions and ii) obviously to satisfy the fault tolerance requirements of the application attached to it. This work is complementary to former research work on the impact of assumption coverage on dependability measures [4].

In summary, a change may have no effect on dependability or require an adaptation of the FTM. In a sense, this is similar to fault injection analysis, an injected fault may have no effect (often called in *No Observation* case in fault injection experiments) or lead to a failure mode (see. [5]). A failure mode requires a corrective action after been diagnosed, an adaptation of the FTM in our case.

Adaptation of FTMs [6] [7] [8] is an important feature for a resilient system as it enables its dependability mechanisms to be adjusted to a new operational context. The aim of *Adaptive Fault Tolerance* (AFT) is to modify the FTMs to comply with assumptions and dependability requirements.

A resilient computing system is a system able to adapt its fault tolerance mechanisms at runtime to comply with its fault-tolerance requirements.

The sources of changes can be multiple as already mentioned in introduction (corrective maintenance, updates and upgrades). The upcoming interest to Agile Development process may have an impact on system dependability, when the rushing implementation phases reduce the validation phase, leading to residual faults. The need for delivering as fast as possible new system versions to the public, the need to perform remote update and upgrade to improve but also sell new features, are de facto economic stakes!

The work reported in the paper addresses the following question: how to estimate the resilience of a system?

III. CHANGE MODEL AND FTM ASSUMPTION

A. Basic Assumptions

We assume in this work that a system host applications that have dependability requirements. After a deep analysis of these dependability requirements, including a *Failure Modes Effects Critical Analysis* (FMECA) or any other risk analysis (e.g. Fault Tree Analysis – FTA), fault tolerance mechanisms are identified.

The *separation of concerns* principle is a prerequisite for adaptive fault tolerance, not only a design time but also at runtime, as illustrated in [8]. An application component (A) is linked to a fault tolerance component (FTM). Many different fault tolerance strategies and corresponding FTMs can be implemented to comply with dependability requirements. All possible solutions with slightly different assumptions can be implemented and loaded into the system. The selection of the appropriate FTM for a given application components can be selected on-line according to monitored assumptions. In practice, according to our system model, an FTM is linked to an application component a priori when the system is put in operation.

B. Adaptation and Change Model

An appropriate *Fault Tolerance Mechanism* (FTM) for a given application depends on several assumptions grouped in three classes: 1) application characteristics (AC); 2) fault model to consider (FM); 3) available resources (AR).

At any point in time, the FTM(s) attached to an application must be consistent with the current values of (AC, FM, AR). These assumptions enable to discriminate FTMs. We denote (AC, FM, AR) the change model.

Several application characteristics (AC) have an impact on the selection of an FTM. In this paper we first consider the following: i) behavioral determinism, ii) application statefulness, iii) state accessibility and iv) fail-silence.

The fault model (FM) considers well-known fault types, e.g., crash faults, omission and transient value faults.

The available resources (AR) play also an important role. Firstly, in the FTM selection, since FTMs require resources to be implemented such as bandwidth, CPU, battery life/energy. This resource criterion may invalidate a solution.

However, resources can be a trigger for FTM change. A lack of resources at a given point in time in the operational life of the system may invalidate an FTM. This implies that a new FTM must be installed according the available resources. This aspect has not been considered in this paper.

Any assumption variation during the service life of the system may invalidate the initial FTMs selected, thus requiring a transition towards a new one. Transitions may be triggered by a new application version with different characteristics or new threats (i.e. fault model change).

A configuration (A ◊ FTM) must remain consistent despite changes in the characteristics of the application and its related fault tolerance requirements.

C. FTM and Assumptions

To illustrate our modeling approach, we consider some conventional fault tolerance mechanisms that will be used as a guiding thread through the remainder of the paper.

Duplex protocols tolerate crash faults using passive (e.g. *Primary-Backup Replication* denoted PBR[2]), or semi-active replication strategies (e.g. *Leader-Follower Replication* denoted LFR[3]). Each replica is considered as a *self-checking*

[2] With PBR, only one replica is active, the Primary. The state of the computation is forwarded to the Backup in a checkpoint. The Backup replica handles checkpoints in normal operation and takes over when the Primary crashes.

[3] With LFR, both replicas are active, the Leader and the Follower. The Leader replies to client's request, while the Follower just executes the request to update its computational state. The Follower takes over when the Leader crashes.

component in both cases. At least 2 independent processors (error confinement areas) are necessary to run these FTMs.

Time Redundancy (TR) tolerates transient faults leading to omission or value errors using repetition of the computation and comparison. This can also be perceived as a way to improve the self-checking nature of a replica.

In this paper, our fault model includes permanent and transient hardware faults or random operating system faults. We do not consider common mode faults.

The above *Duplex strategies* can be combined with *TR* to tolerate both transient and permanent physical faults.

Assumptions / FTM		PBR	LFR	TR
Fault Model (FT)	Crash	✓	✓	
	Omission			✓
	Transient			✓
Application Characteristics (AC)	Deterministic		✓	✓
	State access	✓		✓
	Fail silent	✓	✓	

TABLE I. ASSUMPTIONS AND FAULT TOLERANCE MECHANISM

The considered FTMs, in terms of fault model and application characteristics, are given in TABLE I. The PBR and LFR techniques tolerate the same fault model (crash), but have different application behavioral assumptions. PBR allows non-determinism of applications because only the Primary computes client requests while LFR only works for deterministic applications as both replicas compute all requests. LFR could tackle non-determinism if all non-deterministic actions can be captured. This is not what we consider in a first step. As mentioned previously, PBR requires state access (if any) for checkpointing application state, while LFR does not require state access. TR also requires state access (if any) to restore the previous state of the computation before repetition of the processing.

D. *Evolution scenarii example and possible transitions*

During the service life of the system, the characteristics of the application or its fault tolerance requirements of assumptions can change.

An application can become non-deterministic when a new version is developed. The fault model can also become more complex, e.g., from crash-only it can become crash-and-value. For instance, the PBR→LFR transition (denoted →) is triggered by a change in application characteristics (e.g. inability to access application state). A transition can occur in both directions, w.r.t assumptions variation. A transition obviously implies an off-line validation of the new configuration (A ◊ FTM).

The PBR→PBR+TR (FTMs composition denoted +) transition is triggered by a change in the considered fault model (e.g. crash-and-value). However, the composition of FTMs is not straightforward and requires a deep analysis of the impact of the first FTM on the second one. This analysis is out of the scope of this work. The composition of FTM must be validated off-line before using it, taking care of possible interferences between FTMs as in [9].

IV. BASIC ANALYSIS OF RESILIENT COMPUTING

In this Section we address the following questions: *How to determine a suitable FTM after a change event in order to maintain system dependability? How to estimate system resilience and what insights can be gained?*

The dependability requirements of an application define the fault model to be considered. This fault model determines the suitable set of FTM for this application. Several FTM can be a solution to the problem, but each of them accepts specific characteristics of the application component. Application characteristics are de facto FTM assumptions.

We define the notion of *Consistency Ratio* (CR) to quantify the capacity of a set of FTM to comply with application characteristics and fault tolerance requirements. For a given set of FTM, a list of application characteristics and a set of faults to be tolerated, the CR corresponds to the proportion of the cases where a solution, i.e. a suitable FTM, is found. Each case can be seen as a cell of a table having application characteristics as rows and type of faults as columns. The content of the cell is one or several FTM or nothing. A change event for a given application corresponds to a jump from one cell in the table to another one. The system remains resilient if an FTM is found in the new cell for this application. The probability of finding a suitable FTM in the cell is directly related to the notion of CR.

The following table summarizes the result of our analysis. In TABLE II. we list in columns all possible combinations of fault tolerance requirements an application can specify. In rows, we have all possible combinations of application characteristics. A cell in the table is full when a solution is found. It is empty when no solution is found with the set of FTM and variants considered/available.

The notation for application characteristics is:

- `DT` for DeTerministic, `!DT` if not.
- `ST` for STateful, `!ST` if stateless.
- `SA` for State Access, `!SA` if not.
- `FS` for Fail Silent, `!FS` if not.

The fault-tolerance requirements notation is as follows:

- `C` for Crash faults, `!C` if not considered.
- `O` for Omission, `!O` if not.
- `V` for Value errors, `!V` if not.

`Light Green FTM` means that at least one FTM with their strict definition given in TABLE I. was found to comply with both application characteristics and fault-tolerance requirements. We ignore the case where no fault tolerance requirements are requested, i.e. `!C,!O,!V`.

`Dark Green FTM` corresponds to variants of the initial set of FTM. For instance, TR0 implements a repetition of the computation but with no comparison at the end. TR0 tolerates omission faults and its combination with PBR tolerates more cases in the table.

Last but not least, the `Sky Blue FTM` represents additional solutions with revised definitions, as explained hereafter.

We considered previously that duplex strategies cannot be applied when the application is not fail-silent. The notion of fail silence is probabilistic and depends on the coverage of the error detection mechanisms integrated within the application. The coverage cannot be 100%. The validation process and the measurements carried out in particular using fault injection enables the error detection coverage to be estimated. It is thus the responsibility of the developer to declare if its application is fail-silent or not according to the measurements obtained. However, in both cases, the application being fail-silent or not (`FS` and `!FS`), as determined by the developer, crash faults can be tolerated with duplex strategies. Even for non fail-silent applications, the developer may consider that crash fault must be tolerated despite some value errors can be observed.

A quick look to the table shows that some characteristics have a strong impact on the *Consistency Ratio*. In particular, faults affecting non-deterministic applications are more difficult to tolerate.

In addition, interestingly, the combination of TR and a duplex strategy (PBR or LFR) improves the situation described above. The application of TR improves the error detection coverage for non fail-silent applications, i.e. tolerance to value errors, and thus improves the fail-silent assumption required by any duplex strategy. This is true for deterministic applications and when the state of the computation (if any) is accessible.

FM → / AC ↓	C !O !V	!C O !V	!C !O V	C O !V	C !O V	!C O V	C O V
!DT,!ST,!SA,!FS	PBR	TRO		PBR, +TRO			
!DT,!ST,!SA,FS	PBR	TRO		PBR, +TRO			
!DT,!ST,SA,!FS	PBR	TRO		PBR, +TRO			
!DT,!ST,SA,FS	PBR	TRO		PBR, +TRO			
!DT,ST,!SA,!FS							
!DT,ST,!SA,FS							
!DT,ST,SA,!FS	PBR	TRO		PBR, +TRO			
!DT,ST,SA,FS	PBR	TRO		PBR, +TRO			
DT,!ST,!SA,!FS	LFR, PBR	TR, TRO	TR	LFR, +TRO	LFR, +TR	TR	LFR, +TR
DT,!ST,!SA,FS	LFR	TR, TRO	TR	LFR, +TR	LFR, +TR	TR	LFR, +TR
DT,!ST,SA,!FS	LFR	TR, TRO	TR	LFR, +TRO	LFR, +TR	TR	LFR, +TR
DT,!ST,SA,FS	LFR	TR, TRO	TR	LFR, +TR	LFR, +TR	TR	LFR, +TR
DT,ST,!SA,!FS	LFR						
DT,ST,!SA,FS	LFR						
DT,ST,SA,!FS	LFR, PBR	TR, TRO	TR	LFR, +TRO	LFR, +TR	TR	LFR, +TR
DT,ST,SA,FS	PBR, LFR	TR, TRO	TR	LFR, +TR	LFR, +TR	TR	LFR, +TR

TABLE II. GLOBAL ANALYSIS OF FAULT TOLERANCE SOLUTIONS.

FTM	Global CR	CR for DT	CR for !DT
Light Green	30%	55%	5%
Dark Green	38%	55%	21%
Sky Blue	55%	89%	32%

TABLE III. SUMMARY OF CR MEASUREMENTS.

The TABLE III. summarizes the results. We have shown that with a very limited extension of the mechanisms and a revision of the assumptions required to apply a given FTM, we can impact drastically the number of cases solved. However, some cases remain unsolved because of two problems that can be summarized as follows: i) non-deterministic application for which our current set of FTM cannot tolerate value error, or ii) non-deterministic or deterministic stateful application whose state is not accessible.

The table can be computed using a simple algorithm and the CR measures is easy to obtain. A sensitivity analysis with respect to application characteristics shows that determinism has a strong side effect on the fault tolerance solutions. A full account of this analysis can be found in [10].

V. AUTOMATED ANALYSIS AND MEASUREMENTS

A. Formal notation

Our objective was to automate this analysis. Based on the previous analysis, we can now define a formal notation to represent an application A and its fault tolerance requirements, from which a suitable FTM can be determined. As a result, a configuration A ◊ FTM is consistent according to the definition given below.

*Definition: The design of a critical application is **consistent if and only if the** FTM assumptions and capabilities match Application characteristics and fault tolerance requirements.*

An application `A` has a set of non-functional characteristics, denoted `(ac`$_i$`), k ∈ [1..N]`, N being the total number of characteristics considered in the model. The `boolean` application characteristics are those used previously: `determinism, statefulness, state access, fail silence.` This list can obviously be extended.

The fault tolerance requirements of an application `A`$_i$ correspond to the types of faults `fm`$_j$ the application `A`$_i$ must tolerate. We use again a `boolean` notation to represent this set of faults, `(fm`$_j$`), j ∈ [1..P]`, P being the total number of possible fault types affecting an application component in the system. Examples of such fault types are those used previously: `value fault, omission fault, crash fault.` This list can also be extended with other fault types.

Two vectors, thus represent an application one for application characteristics, one for the fault types that must be tolerated. The FTM attached to the application in a configuration must tolerate such types of faults and be valid for the application characteristics.

$$A = \left(\begin{pmatrix} ac_1 \\ ac_2 \\ \dots \\ ac_N \end{pmatrix}, \begin{pmatrix} fm_1 \\ fm_2 \\ \dots \\ fm_P \end{pmatrix} \right)$$

The vector (ac_i) represents application characteristics. The vector (fm_j) represents the fault tolerance requirements.

The objective now is to determine FTMs making the configuration $A \lozenge FTM$ consistent. As shown in the previous section, an FTM provides a solution to tolerate some types of faults, but its validity depends on application characteristics.

Our final aim is to compute the tables automatically, from the application model given above and a formal definition of the validity of fault tolerance mechanisms.

Let's examine the simple set of mechanisms we have considered previously {PBR, LFR, TR} with their strict definition. Any of these FTM relies on assumptions to tolerate a given type of fault. As an example, the assumptions for the use of PBR are `state access` if the application is `stateful` and `fail silent` behavior. When such assumptions are strictly true, then PBR tolerates the `crash` of the application. The same reasoning applies to other FTM and their combination.

B. FTM assumptions and properties

The assumptions for each mechanism in the set of FTM with respect to application characteristics can be defined by logical assertions. We first use assertions to test the ***compatibility*** of an FTM with the application characteristics. Assertions for the FTM considered are defined as follows:

Assumption for PBR:	`FS and !(ST and !SA)`
Assumption for LFR:	`DT and FS`
Assumption for TR:	`DT and !(ST and !SA)`

The assertion for the combination of several FTM can be deduced from the above logical expressions, such as:

Assumptions for LFR+TR:	`FS and DT and !(ST and !SA)`

The above assertion is stronger than needed; if it is true we guaranty that the FTMs composition is valid. However, a composition may require reduced assumptions. For instance, applying TR implies that fail silence (`FS`) is no longer necessary since TR is a way to improve the fail silence. This is not considered in our algorithms, but reducing the assumption set could improve the CR.

The Boolean expressions allow us to take in account the dependencies between some characteristics. For instance, the state characteristics has an impact on the CR only when the application is stateful. We use assertions to test the ***adequacy*** of an FTM with respect to fault tolerance requirements. Assertions for the FTM considered are defined as follows:

Fault model for PBR:	`!O and !V`
Fault model for LFR:	`!O and !V`
Fault model for TR:	`!C`

Now, we can define properties to check the consistency of any application configuration *(A ◊ FTM)*.

*Definition of **compatibility**: FTM is compatible with A **if and only if** FTM assumptions comply with the application characteristics of A.*

*Definition of **adequacy**: FTM is adequate with A **if and only if** FTM tolerates the fault model required by A.*

*Definition of **consistency**: A configuration (A ◊ FTM) is consistent **if and only if** it complies with both the compatibility and adequacy properties.*

C. Notion of Consistency Ratio

During its lifetime, the several versions of a given application developed and loaded into the system may have an impact on the application characteristics and thus invalidate the FTM that has been attached to the application A in a first place. To help quantify the resilience offered by a set of FTM, we define a notion of *Consistency Ratio*.

*More formal definition of **Consistency Ratio**: For all combinations of application characteristics and fault tolerance requirements, the CR is the ratio of consistent configurations (A ◊ FTM) we can obtain for a given set of FTM.*

This is exactly what we have done manually in Section IV. The two questions we want to address now are:

- Can we compute the configuration tables from the application model and the FTM logical expressions?
- How to compute the CR for such configuration tables and FTM definitions?

D. Computing configuration tables and Consistency Ratio

The proposed algorithm creates the tables from the application characteristics and fault tolerance requirements using the logical assertions established for each FTM in the set of FTMs. From the tables obtained, it computes the CR.

All application characteristics and fault tolerance requirements are encoded with `boolean` values. The application is represented by two `boolean` vectors. For instance, a deterministic (`DT`), stateful (`ST`), with accessible state (`SA`), fail silent (`FS`) application requiring tolerance to crash faults is represented like this:

$$A = \left(\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \right)$$

Each FTM assertion is implemented as a function returning a `boolean` value. The output is `1` if the application *A* matches the FTM assumptions and fault model, `0` otherwise. The assertion for the PBR mechanism is:

```
FS and !(ST and !SA)) and (!O and !V)
```

```
INPUT: Application model, FTMs set
FOR each application characteristics SA
   FOR each fault model FM
      FOR each FTM
         IF (AC,FM) ◊ FTM is consistent THEN
            Store FTM in the cell (AC,FM)
         END IF
      END FOR
      IF the cell (AC,FM) is not empty
         Increment NbrOfConsCells
      END IF
   END FOR
END FOR
RETURN: CR=NbrOfConsCells/TotNbrOfCells
```

Fig. 1. Algorithm for CR computation

Fig. 1 gives the pseudo-code our simple algorithm to automate the computation of the table and the CR. This algorithm was validated using TABLE II. as an oracle. Using the same application characteristics, fault tolerance requirements and FTMs we are able to compute the CR values given in TABLE III. The approach can easily be extended to new characteristics, fault model, and set of FTM.

VI. SENSITIVITY ANALYSIS

This section proposes a sensitivity analysis of the CR offered by a given set of FTM. The aim is to identify application characteristics and types of faults that have the most impact on the CR. We use the algorithm presented in the previous Section to measure the CR when fixing some of the parameters. First, we will address the sensitivity regarding application characteristics (Section A) and then, we will discuss the sensitivity to fault types (Section. B).

This analysis will help us to find the most efficient way to improve the *Consistency Ratio*.

A. Sensitivity to application characteristics

In this Section, we analyse the impact of application characteristics on the CR. For each application characteristic, we measure the CR when the characteristic is set to a particular value, and the other one can vary. In Fig. 2 each bar corresponds to a CR. The X-axis represents each specific characteristic and its assigned value (1 or 0). When a given characteristic is set to 1, it means that it is true for the application. For example, the bar "DT=1" represents the CR offered to an application which will always be deterministic regardless of future updates.

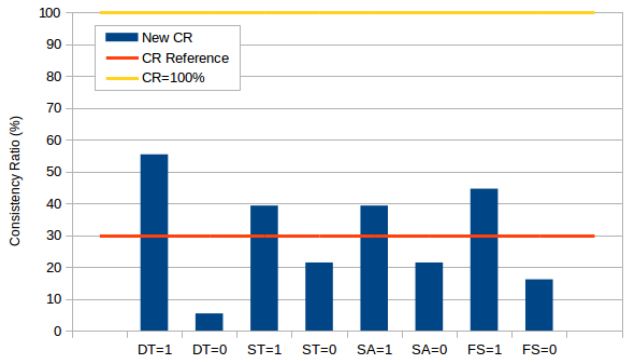

Fig. 2. Sensitivity to application characteristics

For this analysis, the set of FTMs is {PBR, LFR, TR}, with their strict definitions. The CR for this set of FTM was about 30% when no characteristic is imposed (see TABLE III.) This CR value is our *Reference* value in our sensitivity analysis. The question now is: *what is the impact on this reference value when fixing one application characteristic for this FTMs set?*

The CR obtained when fixing a characteristic to 1 is higher that the reference CR value. For instance, when the application is guaranteed to be always deterministic (DT=1) the CR value is improved from 30% to 55%. As a consequence, we can anticipate that when the component is not deterministic, the CR value is lower. All this is confirmed by the results obtained in Fig. 2 where we show the impact of each characteristic. In our analysis, with the set of FTM we consider, we observed that determinism has the most important impact on the CR value. Other characteristics have also an impact, but it is less significant.

The conclusion is that improving the CR implies focussing on this characteristic first. Two solutions are possible to improve it: Either we force the component to be deterministic or we need to include new FTM compliant with non-deterministic applications.

B. Sensitivity to fault types

In this Section, we analyse the impact of the fault types to the CR. The following analysis is done with the extended set of FTMs considered in TABLE II. The reference is 55% in this case (see TABLE III.). We use the same method as previously: when a fault type is set to 1, it is part of the fault tolerance requirements of the application. When it is set to 0, the fault type is not part of the faults considered.

The results are given in Fig. 3. This figure shows the CR variation when a type of fault is removed (i.e. set to 0). As shown, the CR can increase or decrease. This enlightens the strong impact of the fault model, positive or negative, on the CR for a given set of FTM.

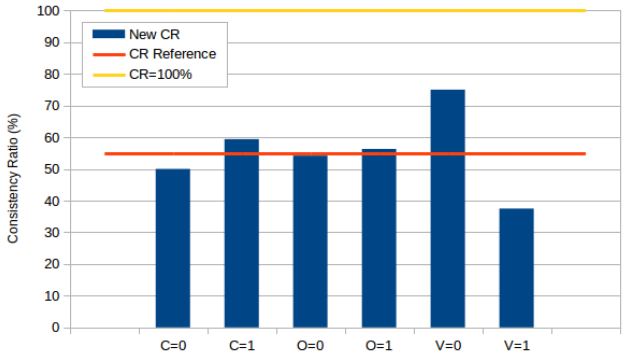

Fig. 3. Sensitivity to fault types

We observe that the CR is increased when we remove a fault type (e.g. value faults removed) from the set of fault types; the reason for this is that this type of fault is not efficiently tolerated with the current set of FTMs, whereas other faults are better tolerated.

Conversely, when the CR is decreased (e.g. crash faults removed) we can infer that these types of faults (namely crash faults here) are better tolerated than other types of faults (value and omission). It is indeed the case as shown previously with the simple set of FTM we consider in this analysis. A first solution to improve the CR could be to extend our set of FTM to include mechanisms dedicated to value faults. Another solution is to work on the application development itself to prevent by construction this kind of faults, e.g. implementing a self-checking applications.

In conclusion, this analysis is a mean to identify the fault types which are badly covered and to find a solution to improve the resilience. Two options are possible: i) extending the FTMs set or ii) removing these faults by

design. The conclusion regarding the impact of application characteristics is similar: i) extending the FTMs set with mechanisms handling more efficiently some application characteristics or ii) removing by design those characteristics that have a bad impact on the *Consistency Ratio*.

VII. Other Resilient Computing Measures

Several other measures can provide interesting insights about the resilience of a system, focusing on its robustness during a given period of time. It is work noting that a lack of mechanism after a change is not a failure. The target application is fragile in the sense that the activation of a fault may lead to a failure since no valid mechanism is active to tolerate that fault. These measures take into account the frequency of changes, the delay to update the affected FTMs, in other words the occurrence rate and the repair rate. In that respect, they are similar to conventional dependability measures, but with a major distinction since we do not speak of failures but fragility!

MTBI – *Mean Time Between Inconsistency*

A sequence of change events during the system lifetime may have no effect, the system remains dependable, or put an application at risk since one or several FTM do not comply anymore with their fundamental assumptions. This measure is similar to the MTBF conventional dependability measure. The main difference is that we do not consider failures, but inconsistencies, so potential risk.

The insights we can gain with this measure is the following: a large MTBI indicates that the various changes in the system do not invalidate FTMs and this is good news. For a given frequency of changes, the larger the MTBI, the better the resilience is.

MTRI – *Mean Time to Repair Inconsistency*

The meaning of repair here refers to the ability to find a solution after a change event, when the current FTM becomes invalid. Either a valid mechanism exists and is loaded into the system, and so the time to repair the inconsistency can be considered null, or it needs to be defined, developed and uploaded at the other end of the spectrum. This measure is similar to the MTTR conventional dependability measure.

The insights we can gain with this measure is the following: a small MTBI indicates that system do not remain fragile during a large period of time in average. In conventional dependable systems, the fault rate (λ) for hardware faults is largely smaller that the repair rate (μ). The change rate (similar to λ) depends very much on the type of systems, so the influence of the update rate (similar to μ) is much higher for a high frequency of changes. Having FTM building block ready that can be combined quickly to solve the problem is of course a way to improve the repair rate.

RE(t) – *Resilient behavior Estimator*

This last estimator corresponds to the proportions of change events at time t that do not lead to an inconsistency. A 100% value would indicate that, despite a sequence of change events since time *0* up to time *t*, no change event led to an inconsistency. The system is perfectly resilient!

VIII. Simulation Tool

As stated before, our approach can be automated, for instance the computation if the Consistency Ratio (CR) can be done with a simple algorithm (see Fig. 1). However, we can go a step further. We have developed a simulator, which combines two aspects of our analysis. The first one is the exhaustive analysis of the Consistency Ratio. The second aspect uses specific scenarios to measure the resilience of the system. A scenario corresponds to a sequence of change events impacting the application profile during a period of time. In our context, an application profile is the core assumptions of application relevant to dependability, i.e. the application characteristics and fault model elements.

The simulator provides the system designer / manager with a tool to analyze configuration choices vs possible evolution scenarios thanks to various measures of resilience. The first objective is to obtain the Consistency Ratio for a set of FTMs and a set of parameters (determinism, state access…). As said in part IV, to compute the CR we need to check for each application profile if there is at least one FTM consistent in our FTMs set.

In this tool we chose to model the consistency property using Boolean expressions as shown in section V.B. We will then evaluate these expressions with the values of `FS`, `SA`... of each profile.

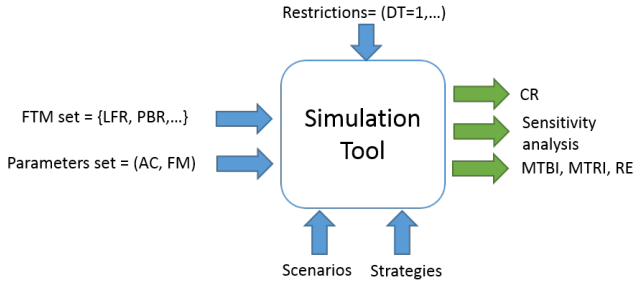

Fig 5. Simulation Tool framework

As you can see on this figure representing the simulation tool, not only the analyzer takes as inputs a set of FTMs and a set of parameters but we can also input some parameters restrictions. These restrictions on the parameters are used to conduct sensitivity analysis to applications characteristics and/or to the faults model. Scenarios and strategies are inputs needed by the simulation tools to measure MTBI, MTRI and RE as described in section VII.

IX. Lessons Learnt and Design Process

The resilient computing interpretation of the CR value refers to the probability of having a compliant FTM after an update of the application. The higher this ratio is, the higher the probability to comply with new application characteristics or fault tolerance requirements. Developing a resilient system requires first the selection of an FTM compliant with fault tolerance requirements and the characteristics of the application. However, as we have

shown, a single FTM may not offer much in terms of resilience. Thus, following the principles of Adaptive Fault Tolerance, some additional FTMs need to be included to future-proof the system. A question then arises: How to select these FTMs?

The design process could be the following: first the system designer needs to list all the FTM that could be implemented and for each one construct all consistency assertions (see Section V.B), respecting the critical assumptions for FTMs. With these elements and thanks to the proposed automated analysis, it is possible to compute the CR value for each subset of these FTM. This exhaustive analysis[4] is illustrated in Fig. 4. We compute the CR value for each of the 2^6 subsets we can construct from the 6 FTM considered previously {PBR, LFR, TR, TR0, PBR+TR, LFR+TR}. Each bar represents the CR value for a given subset and for readability reasons we have sorted the results in increasing order (i.e. the X-axis correspond to the list of 2^6 subsets ranked according to their corresponding CR value).

Then the designer can find all the subsets of FTM that satisfy some CR requirement. Finally, among all these subsets of FTM, the designer must use other criteria, such as required resources or development cost, to select the right set of FTM to implement. It is noteworthy that deciding the right CR requirement is obviously a complex problem in itself and it should at least take into consideration the criticality of the application, its envisioned frequency of updates, and the confidence in the initial fault model.

If no subset of FTM is acceptable with respect to the desired CR, for example because implementation is too expensive, the sensitivity analysis can guide the designer towards other solutions as shown in section VI.

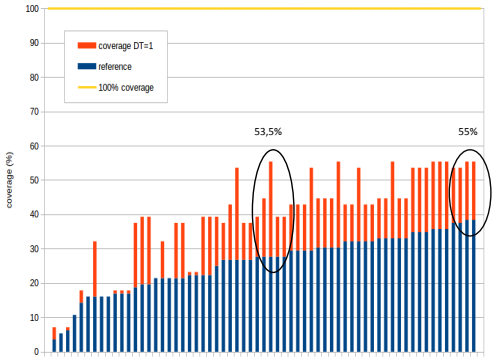

Fig. 4. Exhaustive CR analysis

Fig. 4 shows the CR values for all subsets of FTMs in blue and when the application is deterministic in red. When all mechanisms are used, i.e., the last bar to the right of the bar chart, we reach a CR of 38% without any restriction on the characteristics of the application while we obtain 55% when the application is deterministic. This is consistent with the figures obtain previously. What is more interesting is that with only 3 FTMs in the set we can reach a CR of 53,5 % as you can see in the middle of the bar chart.

We clearly see then that making such an assumption offers the designer more choices. Such insights have an impact on the development of an application in a resilient system. Forcing determinism implies a development discipline, the identification of non-deterministic decisions, and also a non-concurrent implementation of the application. Lastly, new complementary FTMs are needed to cope with non-deterministic decisions, if any.

X. CONCLUSION

In this paper, we proposed an approach for the analysis of resilient computing that requires a deep understanding of the application charateristics, the fault model and the core assumptions of fault tolerance mechanims. We introduced the notion of *Consistency Ratio* as an estimator of the system resilience, i.e. a measure of the system capability to remain dependable when facing changes, but also other measures. This approach is generic in the sense that it can be extended to any application characteristic, fault model, or FTM. The proposed sensitivity analysis and simulator are a mean to manage the development of resilient computing systems and are envisoned as tools to help system designers to take appropriate decisions regarding resilience.

REFERENCES

[1] J.-C. Laprie, “From Dependability to Resilience,” in Proc. of the 38th IEEE/IFIP International Conf. on Dependable Systems and Networks (DSN), suplemental volume, 2008.

[2] M. Albanese, S. Jajodia, R. Jhawa and V. Piuri, "Dependable and Resilient Cloud Computing," in Proc. of 2016 IEEE Symposium on Service-Oriented System Engineering (SOSE), Oxford, 2016, pp. 3-3.

[3] Wiesmann M, Pedone F, Schiper A, Kemme B, Alonso G. Understanding replication in databases and distributed systems., *2000.* in Proc. of. 20th International Conference on Distributed Computing Systems, 2000; pp. 464–474.

[4] Powell D. "Failure mode assumptions and assumption coverage". Fault-Tolerant Computing, 1992. in Proc. of FTCS-22, Twenty-Second International Symposium on Fault Tolerant Computing, 1992; pp. 386–395

[5] Mei-Chen Hsueh ,T.K. Tsai, R.K. Iyer, "Fault injection techniques and tools", Computer, Vol. 30, issue 4, 1997.

[6] K. H. K. Kim and T. F. Lawrence, “Adaptive Fault Tolerance: Issues and Approaches,” in Proc. of the Second IEEE Workshop on Future Trends of Distributed Computing Systems. IEEE, 1990, pp. 38–46.

[7] C. Krishna and I. Koren, “Adaptive Fault-Tolerance for Cyber-Physical Systems,” in IEEE International Conference on Computing, Networking and Communications (ICNC), 2013, pp. 310–314.

[8] M. Stoicescu, J.-C. Fabre, M. Roy, "Architecting Resilient Computing Systems: A Component-Based Approach For Adaptive Fault Tolerance", Journal Of Systems Architecture, Elsevier Eds, Ref. Jsa-D-16-00131R1, Nov. 2016

[9] J. Lauret, JC.Fabre, H.Waeselynck,, “Fine-Grained Implementation of Fault-Tolerance Mechanisms with AOP: To what Extent”, SAFECOMP 2013, Toulouse (F), Sept.2013.

[10] W.Excoffon, J.C.Fabre, M. Lauer, "Analysis of Adaptive Fault Tolerance for Resilienc Computing", in Proc. of the European Dependable Computing Conference (EDCC2017), Geneva, CH, sept.2017.

[4] Although this approach is exhaustive, we avoid scalability issues because our approach is solely focused on the critical assumptions and requirements for AFT.