Redundancy + Reconfigurability = Recoverability

Simon Monkman[1], **and Igor Schagaev**[2]
[1] ITACS Ltd, 157 Shephall View, Stevenage, SG1 1RR, England
[2]Faculty of Computing, London Metropolitan University, 166-220 Holloway Road, London, N7 8DB, England

Abstract - *An approach to consider computers and connected computer systems using structural, time and information redundancies is proposed. An application of redundancy for reconfigurability and recoverability of computer and connected computer systems is discussed, gaining performance, reliability and power-saving in operation. A paradigm of recoverability is introduced and, if followed, shifts connected computer systems toward real-time applications. Use of redundancy for connected computers is analysed in terms of recoverability, where two supportive algorithms of forward and backward tracing are proposed and explained. As an example, growth of mission reliability is formulated.*

Keywords: redundancy; reconfigurability; recoverability; performance-reliability-energy-wise systems

1 Why Recoverability: Instead of Introduction

The human world evolves and progresses by applying knowledge derived from observations of and familiarity with repeatable aspects of nature. Our perceptions, understanding, and ability to model reality enables us to develop the policies, processes, and products required, in order to attempt to control the behaviour of natural phenomena, or human-made objects.

Nature tends to achieve stable and reliable progress (sustainable growth) and avoid regression and degradation. Sustainable growth can be considered as a fundamental descriptor of living matter, while regression and degradation are descriptors of dead matter. A clear differentiation between live and dead is required, but, so far, there has been no substantial research, or projects, on it.

The authors of this paper believe that the fundamental distinction and difference between living processes and dead matter is *recoverability*.

Essentially, recoverability in the system is based on the ability to use available redundancy to recover from environmental, or internal impacts and shocks. Two things are worth mentioning here: first—redundancy is necessary for recoverability, and second—redundancy must be deliberately introduced into systems, policies, and processes to make them resilient and efficient.

The recoverability approach and its analysis, application, and conceptual development in the domain of computers is one of the aims of this paper. The second aim is the analysis of the phases required for the implementation of recoverability for stand-alone and connected computers.

Usually, connected computer systems display fluctuations due to changes in the underlying systems. Reasons for this may include, for instance, workload, software completeness, consistency, and size of applications, or changes and shocks emanating from their environment. So far, networks sporadically and inconsistently exploit recoverability phenomena to tolerate these various fluctuations.

Connected computer (further CC) systems can be considered in terms of time, i.e., as a process of operation. Recoverability can then be applied to keep this process within a restricted set of properties, "smoothing" the process. We can apply and investigate various recovery algorithms implicit in such systems and tune the underlying parameters, reducing the extent of fluctuations and hence, reducing the cost they impose in structure, information, or performance.

Natural recoverability phenomena exist in almost any natural system, but we do not understand them. Hence, we cannot specify how the algorithm works and therefore, use it properly. This is exactly the purpose of the methodology proposed in this paper. In a practical sense, an understanding of recoverability enables us to advocate for the re-design of the whole world of CC systems, making them resilient to internal and external fluctuations.

1.1 Why Reconfigurability: An Example

Let us consider a case: an element is deformed by environmental impact. Destructive deformation of the element could cause the loss of its properties.

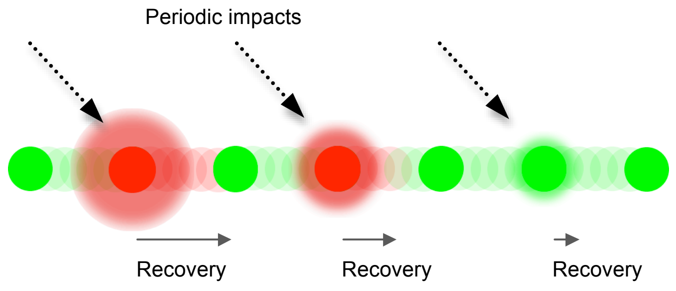
Let's assume an element has internal structural resources (redundancy). Redundancy of the element structure might enable the element to return to its previous state, or condition, after impact. The external impact does not change the element, if redundancy is applied and sufficient. A second impact might occur and be tolerated in exactly the same way. Let us now consider a situation where the element has properties of being alive, such as amoeba.

If an amoeba has sufficient resources available to it to use and to protect itself from the destructive energy of the environment or an impact, it will recover and continue to live—the amoeba exhibits redundancy in order to survive.

If the external event is repeated, the amoeba can self-tune and be able to react to the impact faster, tolerate the event for longer, and as a consequence, suffer less long-term damage. The external event itself might be periodic heat from the sun, cold water, fire or gas, electric discharge, etc.

Having sufficient internal redundancy to tolerate repeated external impacts caused by various events makes recovery possible. Live matter differs from man-made systems in terms of the time required for recovery and the use of available redundancy. The speed of recovery increases when the impact is the same. Here, "recovery training" takes place and either the level of redundancy, or speed of recovery, or both increase. A sequence of impacts and element recovery is presented in Figure 1.

Figure 1. Periodic impacts, element's time to recover



The circles show the state of an element over time, where green indicates an element in a good, or acceptable steady state and red indicates an element under recovery. Figure 1 indicates an element that adapts to the periodic external stimulus, can decrease the time for its recovery.

Where the element may be considered alive, such as in the case of the amoeba, using redundancy for recovery can reduce the time it takes to react to the same event, provided the event is periodic. Thus, life might be defined as the following:

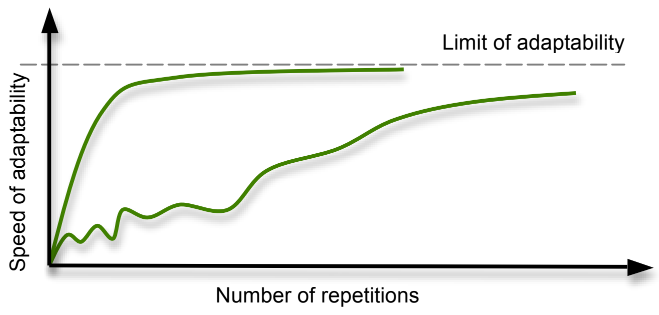
An element is called alive, if in repeatable conditions, it is able to recover progressively, using internal redundancy actively.

The adaptability of the live element has its limits. Figure 2 shows, for example, an element approaching the limit of its adaptability and the role of its ability to recover. Whilst wildlife evolution may be seen as similar to the lower curve, the evolution of "smart" species should be smoother and faster to reach the same, or higher, limit of adaptability—the shortened curve in the diagram.

Thus, our design of information processing systems, computers, especially complex systems such as connected computers, can be measured in terms of efficiency of recovery/resilience in comparison with wildlife phenomena, where available redundancy is used and adaptability grows. In other words, how good we are at designing our systems to be adaptable can be checked against living objects.

What is the point of this? Without external repeatability of events, evolution is hardly possible; having internal redundancy to recover is not enough. Evolution depends on the repetition of the same external events—*i.e., no repetition, no evolution.*

Figure 2. The adaptability of a live element to a repeated external stimulus has its own limits



It means, for example, that the merit of sending a NASA probe searching for advanced forms of life on asteroids is worth questioning. An asteroid does not have the repeatability of environmental events during its flight. Even if life forms were there initially, their redundancy was spent for nothing in attempting to tolerate sporadic impacts.

1.2 Organization of the paper

How is this two-part introduction about recoverability and reconfigurability related to CC systems? At first, nature-made living systems are much more reliable and resilient than human-made ones. Therefore, some of the key principles of "mother nature designs" are good to adapt for CCs. Secondly, an analysis of existing technologies and applications, even if it is brief one, might highlight what is required to make our designs smarter.

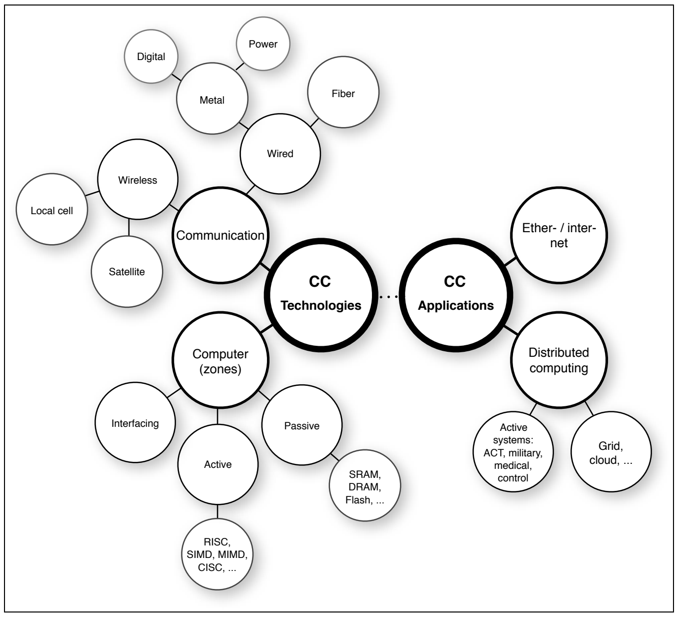
Further, commercially and technologically speaking, we will address recoverability and other properties that might be required for connected computer systems. Why do we need to make this clear? Market segmentation in computer and CC systems might be reduced, or eliminated, enabling unified and modernized technologies to be applied.

We will discuss properties such as reliability and some ways to achieve it, using deliberate redundancy and recoverability when required. We extend redundancy from fault tolerance to PRE-smart system design. PRE here stands for Performance-, Reliability-, and Energy-smart systems. Later, we will be able to estimate the efficiency of redundancy use for reconfigurability and recoverability for CC systems, balancing the trade-off between PRE properties.

2 Connected computers: technologies and applications

CC technologies in general are divided into two almost independent clusters: Communication and Computer, as Figure 3 shows.

Figure 3. Connected computers - technologies and applications



The Communication cluster deals with various media (wired, wireless) using different signal carriers (copper, fibre, air). The cluster faces problems of complexity and the volume of data that needs to be transferred, together with requirements of timely data delivery over complex interconnected networks.

The Computer cluster addresses all three zones of information processing to make them faster and technologically feasible. The zones differ semantically. The Active zone is the one where information is transformed and is currently known: in the form of complex instruction set computers (CISC), reduced instruction set computers (RISC), single instruction multiple data (SIMD) architecture and multiple instructions multiple data (MIMD) architectures. Flynn's [1] classification was used to reference these architectures. The Passive zone is known in the form of static and dynamic memories, flash memory, disks, etc. The Interfacing zone deals with data transfer between zones and getting them in and out from environment.

Historically, computer systems were not really fit for purpose for working within CC systems, which reduces our expectations when addressing the aspect of distributed computing by design. Attempts, such as a transputer, also prove that introducing distributiveness into CC is challenging and not an easy task. CC systems such as the Internet and Ethernet are expanding enormously in terms of data transfer, video, audio and e-mails and are moving in a strange direction, allowing home-makers, young people, financial sector operatives and bureaucrats communicate and "deliver their messages and instructions".

All of the aforementioned applications are not critical in terms of real-time operation; VOIP requires some traffic shaping to deliver packages with time and other constraints, known as Quality of Service. This is what the vast majority of CC systems are using. At the moment, according to various sources, around two billion IP addresses are allocated permanently. This prodigious amount of data requires handling procedures that need to be much more effective, as everyday life becomes dependent on the "health" of CC systems. There is a visible shift in the distributed computing paradigm (using distributed, connected computers to solve large-scale tasks), toward distributed databases, financial services such as ATM, and so-called "cloud computing". Putting scepticism aside and leaving other papers and researchers to discuss what is the real technological progress of cloud computing, we note here only that the efficiency of large-scale applications, including cloud computing, depends on the algorithmic skeleton—graphs of data, control and address dependencies [4] and their use, in order to prepare flexible, reconfigurable and resource-efficient algorithms for distributed computing.

To be effective, distributed computing requires a periodic "tuning" of the CC topology and computers as the elements in that topology. These tunings of application software, system software, topology, and internal structure of the computers should be handled statically, before execution and supported dynamically, during execution.

So far, there has been no visible progress in this direction, in spite of substantial investment under the flag of cloud computing. At the same time, there is a segment of human life that really requires attention and the involvement of CC: safety-critical, real-time active control systems, military applications, health monitoring, etc. All these applications should benefit from CC, but they require the integrity of a CC system, in terms of hardware, system and application software, user and system data, and the billions of connected computers to be applied much more efficiently, following the maxima:

Remark 1. *Technology must help people to become better, not to be more comfortable.*

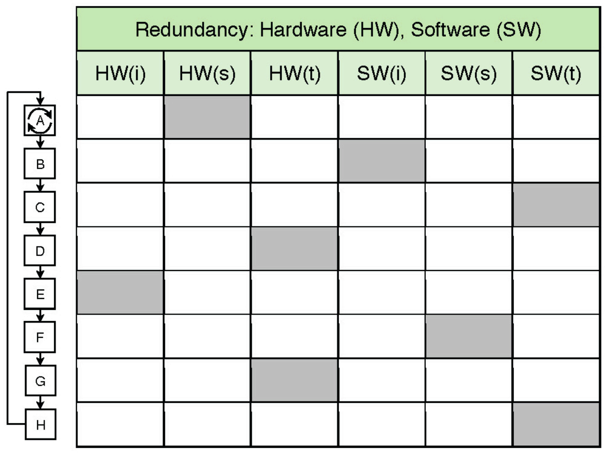
Therefore, safety critical applications (military, health monitoring, emergency management, air-traffic control, traffic control at large) should emerge and exploit existing connected computers. Two approaches to making CC useful are becoming obvious: the application of existing CC to wider and more challenging areas and the use of specially-built, safety-critical systems for "common" applications, as a part of the family of CC.

Ignoring any of theses approaches will lead to bigger market clustering and industry segmentation, resulting in the communication between entities becoming less efficient and which contributes to increased energy and ecological overheads - an unforgivable waste of resources for human race.

2.1 Problems and properties

To avoid this segmentation in technology and market clustering a CC system should be redesigned to have new properties. In addition to the requirement for trustworthy CCs (security of hardware, system and application software and user data), widening CC adoption in terms of application use requires the development of *recoverability*. *Recoverability* requires an implementation of a generalized algorithm of fault tolerance (GAFT). Note also that recoverability is practical, if it is invisible for the application software. GAFT assumes the execution of several sequential steps related to hardware (HW) and software (SW), in terms of proving the integrity of the system, (step A), detection of a fault and determination of its type (step B), defining the "level of damages" Permanent of malfunction (step C), location of faulty element (step D) and reconfiguration of the hardware (step E) and proof of correctness of integrity of software (step F) and determination of correct state (step G) and software to correct in order to continue operation. GAFT has two main phases - one for hardware (steps A-E), another for software (steps F and G). GAFT is initiated if a fault of CC, or any other deviation, has been detected. During the first step, it recognizes fault type in order to gauge location and tolerance.

Figure 4. Redundancy application for GAFT



	Redundancy: Hardware (HW), Software (SW)					
	HW(i)	HW(s)	HW(t)	SW(i)	SW(s)	SW(t)
A		▨				
B				▨		
C						▨
D			▨			
E	▨					
F					▨	
G			▨			
H						▨

As Figure 4 shows the redundancy types application for fault tolerance are based on the categories of *structure* "*s*", *information* "*i*" and *time* "*t*". The white boxes show a possible application of fault tolerance, using the described redundancies.

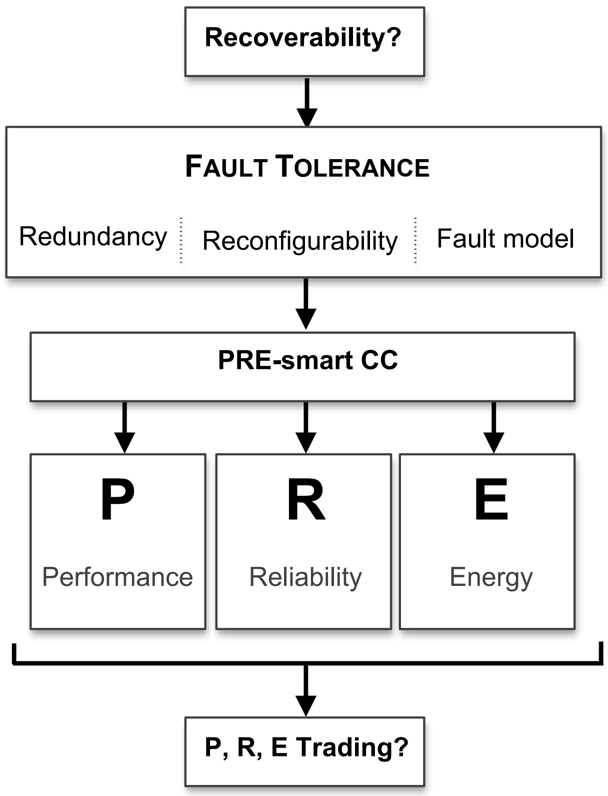
While we are capable of using redundancy for checking, reconfiguration and recovery within a CC system, we should ask ourselves:

Could we use this redundancy for other purposes?

Introducing system redundancy might allow us to achieve recoverability. We need all the ingredients - redundancy, reconfigurability and fault modelling - in order to understand and analyse existing mutual dependencies at every stage of the design and development process.

At the same time, redundancy can be used for reconfiguration of the CC system for other purposes such as performance improvement, or power efficiency. Figure 5 illustrates how properties may be inherited for PRE-wise systems. Thus, PRE-wise systems might be designed rigorously, using reconfigurability and recoverability as system features, if they are introduced at conceptual level. The success of PRE designs for CC systems depends on the careful balancing, or "trading-off", of redundancy against the desired PRE property.

Figure 5. Redundancy and reconfiguration application for PRE systems



2.2 Trading P, R, E

Structure, *Information* and *Time*, as the various types of redundancy, might be weighted, say, in units or values, with or without reference to the steps of GAFT, or any other algorithm where redundancy has been applied to achieve performance-, reliability- or energy-wise features. The relative importance (and cost) of the redundancy type chosen for the steps in the algorithms shown might be introduced as a coefficient α_i, related to the cell i (Figure 4). Similar "valuations" of redundancy types might be applied for any other algorithms designed for the implementation of PRE properties.

While time and information is understandable in units - seconds and bits, the structure, especially structural redundancy requires some extra effort. Note also that time, information and structure are considered as independent variables. Structural redundancy for our purposes might be measured using the graph-related notation:

$$dS: \; < dV, dE >$$

where *dS* denotes introduced structural redundancy, while *dV* and *dE* denote extra vertices and edges added into the structure in order to implement the steps of GAFT, or any other algorithm.

Then, our efforts toward the goal of PRE can be measured quantitatively, as a vector of redundancy use:

$$dR = < dT, dS, dI >$$

In determining the cost of each type of redundancy used and describing the steps of an algorithm to achieve performance-, reliability- or energy-wise improvement, we can quantify each solution, according to the redundancy types applied.

This approach explains and quantifies, for example, the limitations of system software-based developments using Java - it will always consume more time, hardware, software and energy to store and process. In other words, we always will waste much more energy than really required.

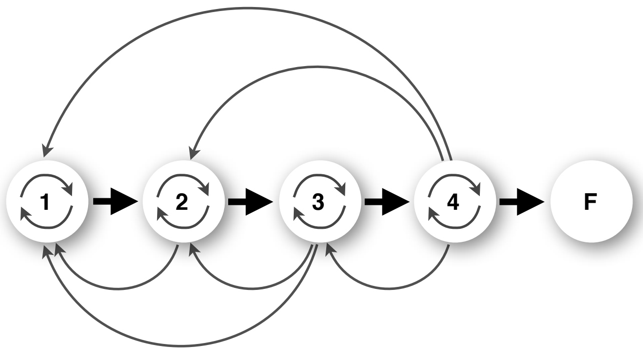
Furthermore, the over-use of flash-based memory will also add to the energy wastage, as the activation of one memory cell in flash requires the application of power to the bulk of a 64K, or 64M memory segment.

The principles of PRE- design should be applied to the CC system as a whole, using the redundancy- and reconfigurability-wise approach for each of the goals. That being the case, tables similar to those proposed above have to be crafted individually for various purposes.

A PRE-wise system design paradigm is the future. When a computer, or CC system is designed with redundancy and reconfigurability in mind, with possible smart configurations and reconfigurations for PRE purposes, the market segmentation of information computer technologies (ICT) will be reduced dramatically. The combination of steps in the sequence described above implementing the declared properties is a simplification, as design of a system is not, in fact, sequential. It most likely follows a pattern as illustrated by Figure 6, where the various steps are dependent on and have feedback loops with other steps.

One approach to cope with these forms of dependencies in the algorithm (or project) phases assumes the application of a semi-Markov model to analyse the impact of these feedbacks on design efficiency [2,5].

Figure 6. Dependencies of project phases



2.3 Recoverability in connected computer systems

Applying the same approach to CC systems to suit real-time and safety-critical applications highlight differences between stand-alone and CC structures:

- Redundancy in CC systems already exists (each computer "deals" with neighbour);
- Latency of threat impact for CC systems is unavoidable;
- Propagation of threat impact for CC systems is similar to flooding.

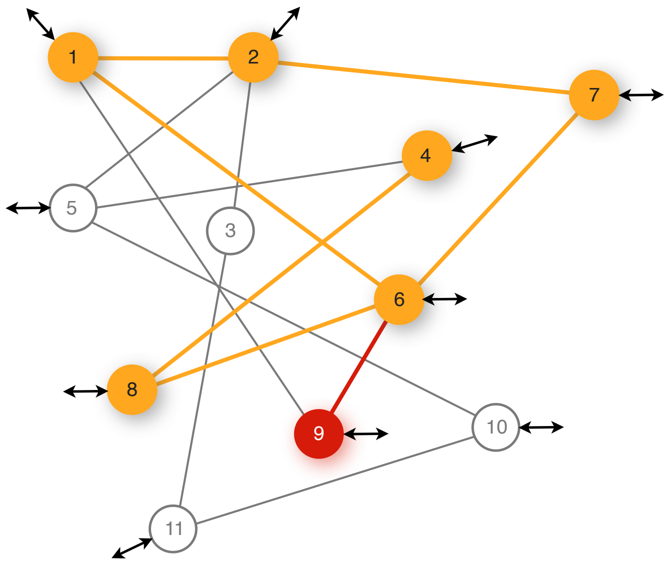
Let us look at a notional segment of a CC topology with incoming and internal connections as Figure 7 presents. Incoming and out-going edges are shown with arrows. Threats here mean physical faults (permanent, or as a malfunction) of hardware, incomplete or deliberately damaged software, viruses, worms, etc. Thus, the recoverability of a CC system might require more effort and extend GAFT actions, namely:

- *Find where threat propagates;*
- *Estimate damages;*
- *Stop propagation;*
- *Find source of the threat (internal, or external);*
- *Exclude, or block the source;*
- *Restore best-fit configuration of hardware;*
- *Restore best-fit configuration of system software;*
- *Restore best-fit configuration of applications.*

To make a system of CC for real-time applications, GAFT must be performed, together with an estimation of the potential consequences for the topology of the CC, as well as its elements. The speed of propagation of a threat through the topology has to be addressed as a factor of performance for recovery.

The potential damages caused as a result of the threat may differ in severity - sometimes substantial and exponentially dangerous (gateway routers), if we do not react accordingly.

Figure 7. Connected computers topology (fragment)



Existing solutions with local restarts and segmental switching do not comply with the requirements of real-time, or safety-critical applications. A CC system can be presented in the form of probabilities of the propagation of a threat (or symptom of a fault) through the topology, where thickness of the edges defines the strength of dependency between vertices. The dependencies between vertices are not symmetrical: vertex 9 might have, say, a much higher impact on vertex 6, than vertex 6 might have on vertex 9.

A propagation of a threat along the CC system might be described as a vector P of predicates {pi} that define the condition for each vertex:

$$P = \left\{ p_1\left(m_1\left(v_1\left(d_1(t)\right)\right)\right), p_2\left(m_2\left(v_2\left(d_2(t)\right)\right)\right), \ldots, p_k\left(m_k\left(v_k\left(d_k(t)\right)\right)\right) \right\} \qquad (1)$$

where *m1,...,mk* stand for models of vertices in terms of vulnerability to threat; *v1,...,vk* are vertices, *d1, ...,dk* are data available about each vertex condition.

Data about each vertex might be accumulated using checking (testing, or online checking, including historic knowledge and their combination), as well as processed in real time.

Note that for a CC system, we assume flood-like threat propagation; i.e. all adjacent vertices to the initial point, namely for vertex 1, one has to consider adjacency with the 2nd, 6th and 9th vertices, vertex 11's adjacency to vertex 3 and 10, etc. The role of the initial point that starts off the process of recovery requires further discussion.

2.4 How this works

The recoverability of CC systems assumes the involvement of two algorithms: Forward Tracing and Backward Tracing. When the symptoms of a threat are manifested through the detection of a change in behaviour at an element, the Tracing algorithm searches through a Dependency Matrix for the subsequent propagation of that threat along the system. The potential consequences to the system can be hereby identified, starting from the vertex from where the threat presence was first detected.

Performing the Forward Tracing algorithm, a cumulative probability is calculated along each possible path (of edges) until a termination threshold ε is reached. Threshold ε is defined empirically using engineering expertise and considered as constant for a particular configuration of a CC.

Another termination condition for searching the path of threat propagation is obvious - checking all dependent vertices. When all elements have been traced, one can fully guarantee 100% threat checking coverage. Unfortunately, this termination condition becomes scale-dependent on CC system size.

Note here that the probabilistic matrix for a system from Figure 7 is not Markovian, because the sum of probabilities on the edges at each node may not be equal to 1; in contrast, several edges of a single node may have significant probabilities.

Figure 8. Forward tracing of possible consequences

```
Algorithm **Tracing** (s, D(N),$D_s$, {Π($p_{s,x}$), x $D_s$ } )
// Input: Dependency matrix D(N) with N elements of a weighted graph G=<V, E>
// Input: The start node s and the reaching node j
// Output:  The set of nodes $D_s$ where x  $D_s$ and $\Pi(p_{s,x}) > \varepsilon$
// Output:  The highest probability $\Pi(p_{s,j})$ of node j reached by node s
Q  // a priority queue based on the higher probability of nodes reached by s
L  // the set of nodes already visited, used to avoid tracing loops
Initialize(Q)  // initialize nodes priority queue to empty
1 For each node v in V do
2   $p_{s,v} \leftarrow \varepsilon$; // set default probability to ε
3   Insert (Q,v,$p_{s,v}$) //initialize the priority queue
4   $p_{s,s} \leftarrow 1$; Increase(Q,s,$p_{s,s}$)  //update priority of s with $p_{s,s}$
5   $D_s \leftarrow$Empty // presume all elements are safe
6   L←Empty
7 for i←0 to N-1 do
8    a*←DeleteMax(Q)  //delete the maximum priority element
9    while $p_{i,a*} > \varepsilon$
10     do
11     Ds←Ds { a*};L←L { a*}; $\Pi(p_{s,a*}) = p_{i,a*}$
12     for every node a in V- Ds- L that is adjacent to a* do
13        if $p_{s,a*} * D_{a*,a} > p_{s,a}$ then
14        $p_{s,a} = p_{s,a*} * D_{a*,a}$;
15        Increase(Q,a,ps,a)
16      end for
16    end while
17 end for
18 Terminate
```

2.5 Probability along the path

In the tracing algorithm, the cumulative probability of threat propagation from one element (vertex) to another along the edges from the suspected node *i* to node *j* (possibly via a series of other nodes), is defined as Π(pi,j).

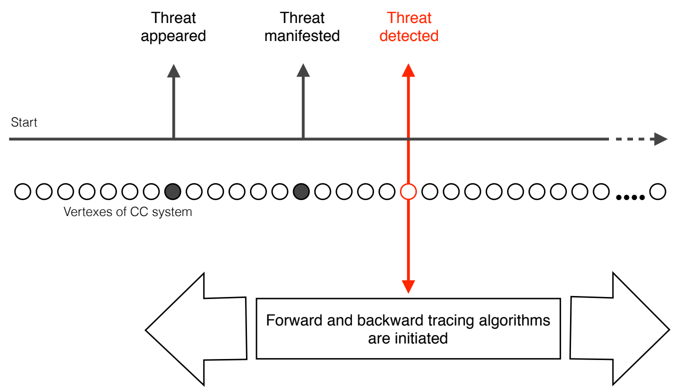
When several paths lead from node *di* to node *dj*, all possible Π(pi,j) are ranked and nodes along the paths are included into the set of suspected nodes. Starting from the vertex, *i*, that manifests the threat, its impact is evaluated by searching from *d1* to all directly, or indirectly connected nodes (elements). The result of this search is a ranked list of the nodes most likely to be affected - the "consequence" of threat propagation. As the threat paths from each node are evaluated, only the edge with the highest probability is followed at each node. At most, each node is only ever included once in any path to ensure termination in a graph which contains loops.

The proposed Forward Tracing algorithm does not solve the problem of threat elimination from CC systems and, at its best, can only be part of the solution. The reason is explained in Figure 9. The time gap between the appearance of a threat at one vertex and the detection of it impact at another has arbitrary duration. Above all, while the consequences are being detected, threat propagation continues. Thus, the Forward Tracing algorithm helps to localize damages, and assist when possible, in order to block propagation, but does not solve the whole problem.

To locate the first damaged node and discover the real reason for its changed behaviour, we need another algorithm called Backward Tracing, Figure 10. This algorithm discovers the source(s), or reason(s) from the sequences of exhibited threat symptoms and defines areas where each element (vertex) was involved. Thus, we search for the reason, not just the symptoms.

When the elements that are likely to be the cause of the manifest discrepancies are detected, the recovery is initiated from the vertex where the threat first appeared dealing with the damaged area only, reducing the need for the brute force of a restart, saving real-time mode for the whole CC system. The results of the recovery process also need to be saved for security improvement, monitoring of reliability and maintenance efficiency.

Figure 9. Threat propagation timing along a CC system



The threat checking procedure over a CC system might be activated, either by a signal indicating that there is a discrepancy in behaviour of one or more elements (vertices), or by a predefined sequence of maintenance, if necessary. For the purpose of maintaining CC system integrity, the procedures for condition checking might be initiated by choosing any vertex of the CC system at random, or even in a loop, covering all vertices, when it is convenient.

Figure 10. Backward threat tracing for a CC system

```
Algorithm **Backward Tracing** ($s$, $R^{(N)}$, $S_s$, $\{\Pi(p_{s,x}), x \in R_s\}$)

// Input:  The Recovery matrix $R^{(N)}$ with N elements of a weighted connected graph G=<V, E>
// Input:  The suspected node s
// Output:  The set of nodes $S_s$ in which $x \in S_s$ and $\Pi(p_{x,s}) > \varepsilon$
// Output:  The highest probability $\Pi(p_{j,s})$ of node j reaching node s

Q // a priority queue based on the higher probability of nodes reaching s
L // a set of node have been visited. It is to avoid loop tracing
Initialize(Q)   //initialize nodes priority queue to empty
1 For every node v in V do
2 $p_{v,s} \leftarrow \varepsilon$;
3 Insert(Q,v,$p_{v,s}$) //initialize the priority queue
4 $p_{s,s} \leftarrow 1$; Increase(Q,s,$p_{s,s}$) //update priority of s with $p_{s,s}$
5 $S_s \leftarrow$Empty //initialize the set of originating node to empty
6 $L \leftarrow$Empty //nodes have been visited is set to empty
7 for $i \leftarrow 0$ to N-1 do
8    $a^* \leftarrow$DeleteMax(Q) //delete the maximum priority element
9    if $p_{a^*,i} > \varepsilon$
      Then
      $S_s \leftarrow S_s \cup \{a^*\}$; $L \leftarrow L \cup \{a^*\}$; $\Pi(p_{a^*,s}) = p_{a^*,i}$;
11       for every node a in $V - S_s - L$ that is adjacent to $a^*$ do
12        if $p_{a^*,s} * R_{a,a^*} > p_{a,s}$
13          $p_{a,s} = p_{a^*,s} * R_{a,a^*}$;
14         Increase(Q,a,$p_{a,s}$);
15    else //correspond to $p_{a^*,i} < \varepsilon$
16      Terminate
```

2.6 How much recoverability costs

As shown above, recoverability requires the introduction of several new processes into CC system management, including online checking of CC conditions and the implementation of two mentioned above algorithms. The gradient of this change is a function of the quality of checking (coverage), success of recovery (algorithms of tracing) and quality of maintenance shifting it to the "light" mode with preventive actions against threats.

The gain from introduced and implemented recoverability was recently measured using a comparison of a standard CC system with a system that implements real-time maintenance was analysed in details in recent book [3].

3 Conclusions and future work

- Recoverability supported by redundancy and reconfigurability is introduced and analysed for connected computer systems.

- A design concept of PRE-wise (Performance-, Reliability- and Energy-wise) systems is proposed as a unified approach.

- Shown that recoverability using Forward and Backward Tracing algorithms makes connected computer systems closer to real-time and safety-critical applications.

- As a future development, it is suggested that the development of a PRE framework, assuming mutual dependencies of phases of design, development and run time use using a semi-Markov model.

4 References

[1] Flynn M. "Some Computer Organizations and Their Effectiveness"; IEEE Trans. Comput., Vol. C-21, 948, 1972.

[2] Birolini A. "Reliability Engineering Theory and Practice." 6th ed., Springer-Verlag: Berlin, Heidelberg, Germany, 2010.

[3] Carbone J., Schagaev I. "Active Conditional Control: Analysis." Applied Cyber-Physical Systems, Springer, ISBN 978-1-4614-7335-0, 2013.

[4] Gutkneht J., Kaegi T., Schagaev I. "ERA: Evolving Reconfigurable Architecture"; SNPD, 11th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, London, UK, 9–11 June 2010.

[5] Plyaskota S., Schagaev I. "Economic Effectiveness Of Fault Tolerance"; Automatic and Remote Control, No. 7, 131-143, 1995.

[6] Schagaev, I.; Kirk, B.; Schagaev, A. "Method and Apparatus for Active Safety Systems"; UK Patent GB 2448351, INT CL: G05 9/02 (2006.01), Granted 21.09.2011.